AD-A037 947 INCO INC MCLEAN VA F/G 9/2

TRANSPARENT INTEGRATED INTELLIGENCE NETWORK QUERY INTERMEDIATE --ETC(U)

JAN 77 P STYGAR, A PUCHRIK, M TUREK F30602-76-C-0090

UNCLASSIFIED RADC-TR-77-39 NL

1 OF 2

ADA037947

FC.

(12)

RADC-TR-77-39
Final Technical Report
January 1977

TRANSPARENT INTEGRATED INTELLIGENCE NETWORK QUERY INTERMEDIATE PROCESSOR

INCO Incorporated

Approved for public release; distribution unlimited.

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public including foreign nations.

This report has been reviewed and is approved for publication.

APPROVED: James W. Bradshaw

JAMES W. BRADSHAW 1st Lt USAF
Project Engineer

APPROVED: Howard Davis

HOWARD DAVIS
Technical Director
Intelligence and Reconnaissance Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-77-39	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): TRANSPARENT INTEGRATED INTELLIGENCE NETWORK QUERY INTERMEDIATE PROCESSOR.		5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. 3 Nov 75 - 2 Nov 76.
		6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Paul Stygar, Andrew Puchrik, Margaret Turek		8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0090 NEW
9. PERFORMING ORGANIZATION NAME AND ADDRESS: INCO INC, 7916 Westpark Drive, McLean VA 22101		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 45941016
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (IRDA), Griffiss AFB NY 13441		12. REPORT DATE: January 1977
		13. NUMBER OF PAGES: 188
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same (12) 181p.		15. SECURITY CLASS. (of this report): UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: 1st Lt James W. Bradshaw (IRDA)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Data Base Transparency
Query Language Transparency
Integrated Computer Network
Data Base Standardization
Distributed Data Bases

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This Final Report presents a functional design and preliminary specifications for a query language translator to provide query language transparency in an intelligence network, so that one query language may be used to access data bases which otherwise would be accessed by different query languages.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

407 275

EVALUATION

This report describes the work performed on defining certain modules necessary to obtain an integrated intelligence network.

One of the problems associated with establishing an intelligence network is that problem which plagues many multi-computer systems. Each computer site has a specialized data base tailored for that site's functions. In order to integrate two sites, some compromise must be made between one extreme of making the two sites completely identical to the other extreme of keeping the sites totally independent.

This report concentrates on one aspect of this multi-computer site integration. It defines, a query language translator module which is tasked with the responsibility of allowing multiple query languages to be used in the network while providing the standardization in query specification the network needs in order for the integration to operate efficiently.

James W. Bradshaw

JAMES W. BRADSHAW, 1st Lt, USAF
Project Engineer

PREFACE

This Final Report presents the system functional and preliminary specifications for the Query Intermediate Processor (QIP) of the Transparent Integrated Intelligence Network (TIIN). This document has been produced by INCO, INC., of McLean, Virginia, under the cognizance of Rome Air Development Center (RADC), Griffiss Air Force Base, Rome, New York, under Contract No. F30602-76-C-0090. The technical work reported herein is based on the transparency concepts formulated by Warren Polk in 1974. The authors were Paul Stygar, Margaret Turek, and Andrew Puchrik. Significant contributions were made by Ted Reiss, Marianne Russek, Lelia Irby, Margaret Mix, Gary Kincaid, Owen Radcliff, Landon Woody, John Voss, and other INCO employees through the design review process.

ABSTRACT

The Transparent Integrated Intelligence Network (TIIN) addresses the need to integrate data residing in distributed data bases throughout the world, making it accessible to users of an extended intelligence network. This report details the progress to date of that portion of the Transparent Integrated Intelligence Network development concerned with host selection and query translation. This effort has been designated as the Query Intermediate Processor (QIP) development.

This report includes functional and preliminary specifications for those data structures and modules concerned with:

- o Translation of a user data request into a common system query format which is independent of host query languages;
- o Selection of the data base(s) which will satisfy the data request;
- o Translation of the host-independent query format into the native query language of the host data base management system (DBMS).

Also included in this report are: an over-all view of the TIIN system and development concepts; host query language syntax descriptions; an account of TIIN validation exercises; a bibliography; and a glossary of terms relevant to the TIIN development effort.

This report should be viewed as one in a projected series of documents describing the overall TIIN development effort. Preceding reports include the TIIN Development Plan (November 1975) which provides an overview of the TIIN effort, and the Transparent Intelligence Language Facility Final Report (RADC Contract Number F30602-75-C-047, June 1976) which deals with the maintenance of a data directory for the TIIN system. Forthcoming reports should detail implementation of the QIP, development of the query distribution subsystem, response normalization, and terminal access procedures.

This report supersedes the Interim Technical Report (August 1976) with the same title.

NR

TABLE OF CONTENTS

TABLE OF CONTENTS (con't.)

TABLE OF CONTENTS (con't.)

TABLE OF CONTENTS (con't.)

TABLE OF CONTENTS (con't.)

TABLE OF CONTENTS (con't.)

*Appendix I intentionally omitted.

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (con't.)

LIST OF ILLUSTRATIONS (con't.)

SECTION I

INTRODUCTION

1. OVERVIEW

This Final Report details the initial system design of the Query Intermediate Processor (QIP) of the Transparent Integrated Intelligence Network (TIIN) under Rome Air Development Center contract number F30602-76-C-0090. This initial design is an integral part of the long-range development of a Transparent Integrated Intelligence Network (TIIN), user-oriented distributed data access and processing capability encompassing world-wide resources. This report supersedes the Interim Technical Report produced in August 1976 for the same contract. A short description of the TIIN system and related background is provided in Appendices F and G of this report. For a complete account of the background of the TIIN/QIP effort dealt with herein, the reader is referred to the INCO publications listed in the bibliography in Appendix C.

2. DEVELOPMENT APPROACH

The TIIN/QIP Project is INCO's second contract effort concerned with the development of the Transparent Integrated Intelligence Network. In the first part of this effort, project personnel studied analyst requirements as they relate to the TIIN design. They have reviewed relevant studies conducted under other government contracts. They have also interviewed people familiar with the intelligence analyst's working environment. Meetings with personnel involved in updating ADP facilities at the National Military Intelligence Center (NMIC) were held. As a result of the above, a list of assumptions were drawn up under which the TIIN development could proceed.

Next, project personnel analyzed the structure of various query languages, including SEAWATCH, TILE, DIAOLS, and TIMS. Results of these analyses were published as internal technical memos.

Then, project personnel developed preliminary design specifications for the QIP modules and data structures involved in query translation and host selection. A design review was performed using a preliminary version of this Final Report. The results of these efforts have been combined to produce this Final Report.

3. PROJECT OBJECTIVES

The tasks and objectives of the TIIN/QIP project are:

- to analyze intelligence data base structures, query languages, and retrieval techniques
- to identify areas of commonality among data base systems which would provide a basis for the TIIN
- to analyze security requirements and human factors involved in the TIIN development
- to develop concepts to serve as a basis for the preliminary design
- to develop an architectural structure for the preliminary design for the QIP subsystem
- to document the preliminary design and its validation
- to integrate the TIIN/QIP preliminary design with existing TOSS architecture and software.

4. RESEARCH AND DEVELOPMENT ACTIVITIES

The following research and development activities have been performed in compliance with the contract:

- analysis of prevalent intelligence data base structures and query languages
- documentation of results of query language syntax analyses
- development of preliminary specifications for the internal data structures to be used during the various stages of query translation
- development of functional and preliminary specifications for the Query Intermediate Processor (QIP) query translation and host selection facility
- delivery of the Interim Technical Report
- design review of the TIIN/QIP specifications
- delivery of the Final Report.

5. UNDERLYING ASSUMPTIONS

The following list of assumptions is included in this overview section to facilitate an understanding of the QIP design. Further discussion of the assumptions and concepts underlying the TIIN/QIP design may be found in Appendix F of this report.

a. DIAOLS as Host Language for TIIN Prototype

Much of the initial TIIN/QIP design effort is predicated on an eventual implementation of a prototype TIIN system after the major subsystems have been designed and specified. The preliminary QIP design has been oriented (in its language dependent aspects and in the host query examples) towards the DIAOLS language, since DIAOLS poses many intricate problems for the QIP module. The study of the problems posed by the TILE query language was limited in scope due to inadequate documentation.

b. QIP Design Based on Hierarchical Files

A review of the prevalent query languages in the intelligence community resulted in the conclusion that most of the files in the intelligence community are hierarchical files with relatively few levels of hierarchy. For the TIIN prototype the set of standard operators will be limited to the basic operators necessary for selection of records in a hierarchical file.

c. Design to Promote File Standardization

File standardization is a complex, on-going process, and thus it is important that the TIIN system include features which promote the further standardization of intelligence files. As a result of this basic assumption the TIIN/QIP design provides for translation of element names and data values at either the user node or the host node or both.

d. Design for Problem-Oriented Data Analysts

Informal discussions and a review of a survey of intelligence analysts have corroborated the viewpoint that a small percentage of data analysts generate a large percentage of the actual usage of an all-source data network. This usage gap might be eliminated by an extensive training program. However, the existence of a usage gap provides a strong motivation for the TIIN effort to simplify data access procedures and to develop transparent processors which do not burden the problem oriented analyst with a myriad of data retrieval rules.

e. Inclusive Language Design

The basic goal of the TIIN/QIP effort is to develop a translator for an "all-host" language, so that the intelligence analyst may learn a single query language to query any host. Any such effort must deal with the myriad of differences between the prevalent host query languages. In many cases the differences are only syntactic and thus relatively trivial for a translator-based approach. In some cases the differences are algebraic; that is, an algebraic transformation can be used to eliminate the difference. In other cases the differences by rewriting the query. For example, geographic searches such as those provided in DIAOLS are not available at some of the other host systems. If the "all-host" query language supported by the TIIN/QIP design were to include only features which could be handled on a uniform basis (equally supported at each host), the collection of uniform features would not include enough features to be useful. Thus, the TIIN/QIP design must support language features on an inclusive basis. The all-host query language must include many language features (e.g., geographic searches) which are not uniformly supported by all hosts.

6. REPORT ORGANIZATION

The following paragraphs summarize the five major sections and the appendices of this report. The appendices contain significant information and should, therefore, be considered as part of the main body of the report. A reader who is not familiar with the TIIN effort might begin with Appendices F, G, and H.

a. Section I: Introduction

The first section provides an overview of the Final Report in relation to the fulfillment of the contract and in relation to the overall TIIN development.

b. Section II: Functional Design

This section provides a description of the functional design of the TIIN/QIP modules and data structures. It describes the partitioning of tasks and directory data between the user node and the host node.

c. Section III: Preliminary Specifications for Data Structures

This section provides preliminary detailed design specifications for the data structures used as interfaces between the modules of the TIIN/QIP subsystem.

d. Section IV: Preliminary Specification of QIP Modules

This section includes decision flow diagrams for the TIIN/QIP modules.

e. Section V: Summary, Implications and Future Development

Section V consists of a discussion of research implications, the alternative courses of future development, and a contract completion summary.

f. Appendix A: Descriptions of Query Languages

This describes the file structures and the syntax of several query languages prevalent in the intelligence community.

g. Appendix B: TIIN Validation Exercises

This illustrates the TIIN/QIP data structures for one sample query in successive processing stages.

h. Appendix C: Bibliography

The bibliography lists research material relevant to the TIIN/QIP design effort.

i. Appendix D: Glossary

A short explanation is provided for the acronyms and terms specific to the TIIN/QIP design.

j. Appenidx E: Tall-Ships File

A short data base is provided to lend substance to the sample queries used in this report.

k. Appendix F: Development Background

This describes the software environment and the transparency concepts underlying the TIIN system.

l. Appendix G: Description of the TIIN System

This comprises a brief functional description of the modules in the TIIN system.

m. Appendix H: Language Transparencies Achievable via TIIN

A variety of features are possible in an "all-host" query language translated by the TIIN system, as described here for a hypothetical Transparency Examples Language (TEL).

n. Appendix J: Multi-File Queries

This appendix describes an approach to solving the problem of multi-file queries in the TIIN system.

SECTION II

FUNCTIONAL DESIGN

1. GENERAL DESCRIPTION OF TIIN/QIP

The Query Intermediate Processor (QIP) is a subsystem of the TIIN system. QIP is concerned primarily with the host-related aspects of the query translation process. These host dependent aspects include host selection, translation of element names and value codes to the names and value codes of the host, and transformation and translation of the query format to conform with the data access rules of the host.

a. The Software Environment of QIP

QIP interfaces with the other subsystems of the TIIN system which is described in Appendix G. One subsystem (TAP) handles all the interaction with the user, and converts the user query to a standard internal format (reverse Polish notation) for delivery to QIP. The element name translation, value code translation, and host selection are performed by QIP based on the information stored in the Network Access Directory (NAD) maintained by the TILF subsystem. The messages and query responses to the user are handled by the RN subsystem which has an information flow which is parallel to and in the opposite direction of the information flow of the QIP subsystem. All TIIN intra-system communication which involves inter-node communication is handled by the QDNC subsystem, which maintains query status information in addition to its function of converting interface data to WICS Common Format (WCF) for data transmission.

All the TIIN subsystems operate under the Terminal Oriented Support System (TOSS). The user interface (TAP) utilizes the Terminal Transparent Display Language (TTDL) of TOSS. The Query Distribution Executive (QDNC) utilizes TISS, the communications interface package of TOSS. The QIP transmits the query to the host DBMS using TOSS facilities.

b. User QIP versus Host QIP

The QIP subsystem consists of a host-independent module called the User QIP and a collection of host-dependent modules called Host QIPs. The User QIP (Figure II-1) converts the query to a user-independent format and selects one or more

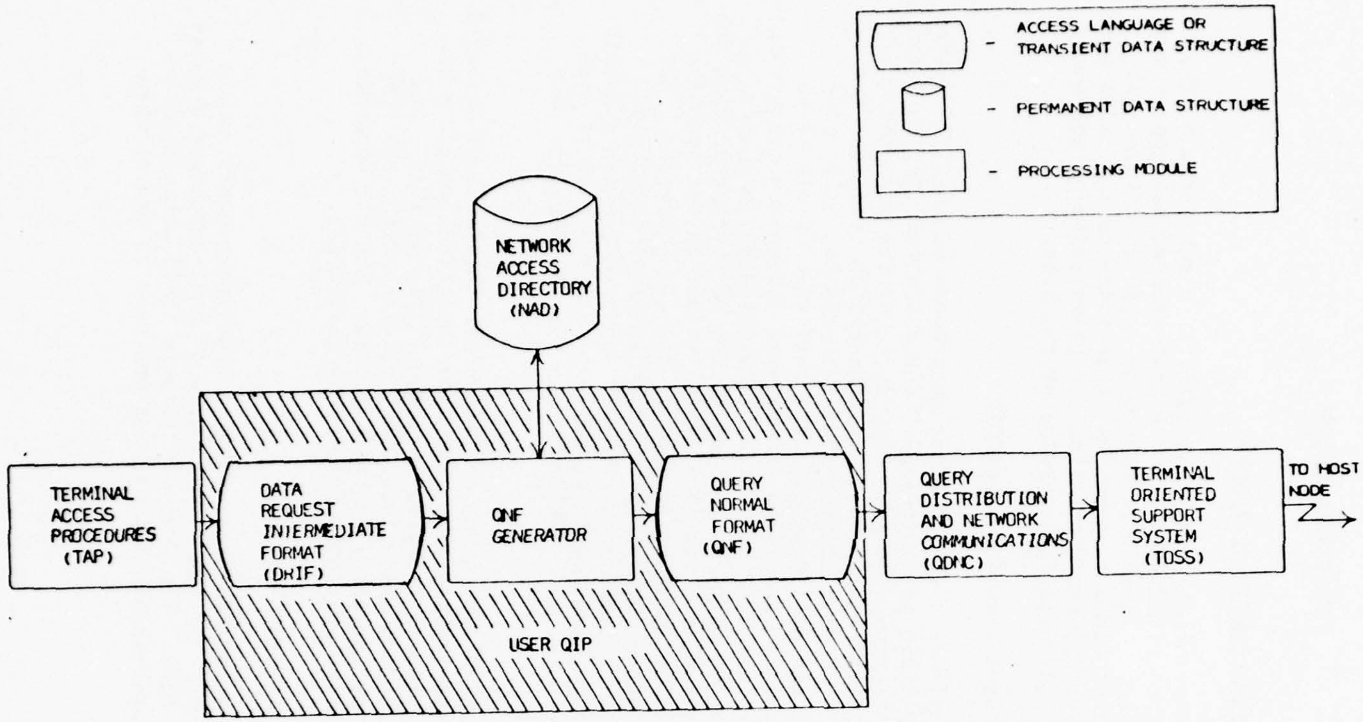


Figure II-1. User QIP Modules and Data Structures.

host(s) to process the query. The Host QIP (Figure II-2) converts the query from the TIIN standard format (Query Normal Format, or QNF) to the host query languages and submits the query to the host for execution.

The User QIP accesses the element information in the User NAD which is that portion of the Network Access Directory which contains the translation tables for user elements versus TIIN standard elements. The Host QIP accesses the file structure information in the Host NAD which is the portion of the Network Access Directory which contains the translation tables for host elements versus TIIN standard elements. The organization of the NAD into User NAD and Host NAD components avoids the update and size problems which would be entailed if every user node and every host node had a complete copy of all the information in the NAD.

The User QIP and Host QIP do not communicate directly. Instead, queries are passed by the User QIP to QDNC at the user node. QDNC at the user node forwards the query to QDNC at the host node by utilizing the TISS communications processor. QNDC at the host node then passes the query to the Host QIP.

c. General Capabilities

QIP enables a user to submit a query in his local query language while addressing any data base in the network. The queries are limited to those retrieving information from a data base. File updating is not being considered as a TIIN capability because it is assumed that the analyst has no need to update on a network-wide basis.

The following lists some of the capabilities provided to the user by QIP.

(1) Query Language Translation

Translation of queries from the user's query language into the host query language will be provided by the TIIN system.

(2) Automatic Selection of Files

The system will supply the appropriate data base and file name corresponding to the data elements mentioned in user queries.

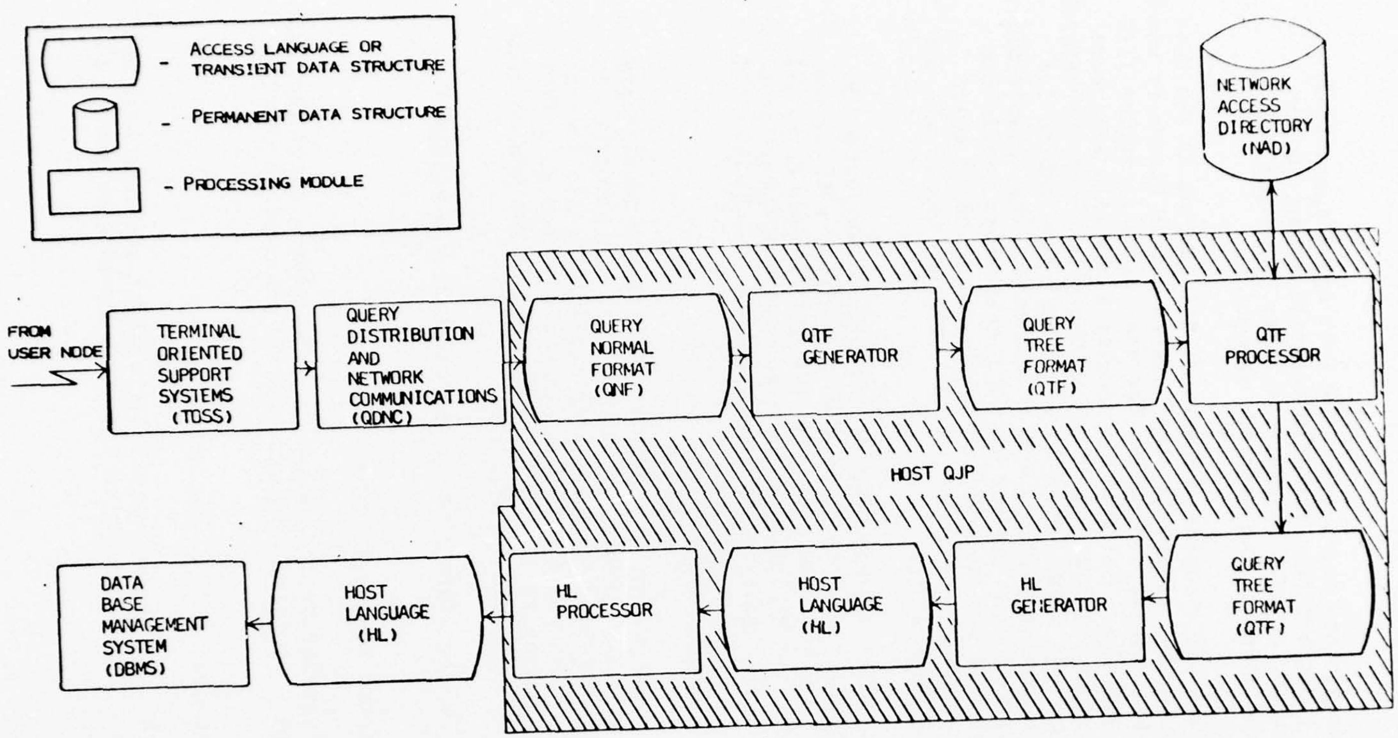


Figure II-2. Host QIP Modules and Data Structures.

(3) Non-Standard DBMS Features

Some special search capabilities (such as geographic searches) which are not available at every host DBMS will be available through TIIN, but only if the capabilities required for the query coincide with the capabilities at whichever DBMS receives the query.

(4) Element Name Translation

The analyst may use element names different from the element names at the host data bases. TIIN will provide translation into other sets of element names existing at different data bases.

(5) Conversion of Values

The various data bases are not uniform in the conventions for representing values; conversion will be required for differing units of qualitative elements, as well as translation among differing code conventions for corresponding value sets of a discrete nature.

d. Operating Environment Assumptions

The QIP design is based on several underlying assumptions, not explicitly stated elsewhere. This section lists several such assumptions to introduce the reader to the environment QIP is designed to operate in.

(1) "All Source" User Access to Data

All users at all nodes have equal access to all elements in the TIIN network. No security is provided by TIIN. The local Data Base Administrator (DBA) may maintain privacy by not defining elements to the NAD (cf. Section II-7d).

(2) One Query Language for All Hosts

Each user will be able to access all files in the network by using a single query language. For the TIIN prototype the user's query language will be the TAP query language. At later development stages it is possible that other query languages will be available to the user, each providing "all host" access.

(3) Network Nodes

Each node in the network will have a NAD, a TIIN system, and, optionally, one or more data bases.

e. Design Objectives

The main objective of the TIIN design is to obtain a simple, open-ended system. The simplicity of the initial design allows for early evaluation of the feasibility of the design and an overview of the problems that need to be solved. The open-ended design will serve as a basis for later augmentation and enhancements of the design.

2. QIP COMPONENT DATA STRUCTURES AND MODULES

a. **Data Request Intermediate Format (DRIF)**

The DRIF is a data structure used to pass queries from Terminal Access Procedures (TAP) to QIP at at the user node. The DRIF is language independent, but not data base independent.

b. Query Normal Format (QNF)

The QNF is a compact data structure used to send queries from the User QIP to the Host QIP. The QNF is fully independent of the host query language and data base.

c. Query Tree Format (QTF)

The QTF is a data structure, used by the QTF Processor at the host node. The QTF is equivalent to the QNF, and allows for easy validation and restructuring of the query.

d. **Network Access Directory (NAD)**

The NAD is a distributed data base containing information about intelligence data bases in the network. Information contained in the NAD allows the User QIP to perform element name translation, value translation, and host selection.

e. User QIP

The QIP at the user node consists of a single module, the QNF Generator. The functions of the QNF Generator include:

- DRIF to QNF conversion

- Element name translation

- Data value translation

- Host selection.

f. Host QIP

It is the function of the QIP at the host node to validate a query given the restrictions imposed by the host query language and data base, and to translate the query into the host query language. The following are the modules comprising the Host QIP:

(1) QTF Generator

The QTF Generator converts a query from QNF to QTF.

(2) QTF Processor

The QTF processor validates and restructures the query, and performs element name and data value translation from the network conventions to the host conventions.

(3) Host Language (HL) Generator

The HL Generator accepts the validated QTF query and converts it into the host DBMS query language.

(4) Host Language (HL) Processor

The HL Processor accepts a character string representing a query in the host language, as generated by the HL Generator, and performs any final adjustments required before submitting the query to the host DBMS.

3. STANDARDIZATION

a. Definition of Standardization

The entire TIIN/QIP design depends on one very important assumption, the assumption that standardization will exist. The standardization required by QIP applies to two areas, element names and data values. Element name standardization, as used in this context, refers to equating data elements in

different files (or in different data bases). Two or more elements can be equated and all such elements are said to contain the same type of information. As an example, the two element names LOCATION-OF-SHIP and SHIP-LOCATION both contain the same information, the location of a ship, but their names are not the same.

Value standardization refers to equivalencing sets of legal values for two or more equated element names. There are many facets to this problem, but in general the values of elements in two different data bases cannot be compared by doing a character-by-character scan. One or both values must be translated into a common value set prior to performing such comparisons. Value translation may consist of right- or left-justifying values, zero or blank filling, converting to a different set of units (i.e., kilometers to miles), or translating one set of coded values into a different set of codes. The last kind of value translation is by far the most prevalent in intelligence data bases.

b. TIIN/QIP Approach to Standardization

In order to solve the standardization problem, the TIIN design assumes that a set of standard network elements will be defined, and that, for each element, a standard set of values shall be defined. Each element name in the network will then be equated to a unique network standard element name, and all legal local values for that element will be mapped into the set of values for the network standard element. This approach has several major advantages over direct translation without using a network standard.

The QNF, containing network standard element names and values, is data base independent and can be routed to any data base without any need to alter its content.

Each name and value existing in a data base only needs to be mapped into one standard name or value, rather than being specifically mapped into all corresponding names and values in other data bases. The number of such mappings is therefore 2*N, rather than N(N-1) required for N direct mappings.

The NAD at each network node needs only to contain information about the local data base, rather than information about all data bases in the network.

c. TIIN/QIP Requirements for Standardization

Several phases of the QIP design depend on the existence of standardization. The following user transparencies cannot be implemented in QIP without element name and data value standardization:

The host selection algorithm depends on a list of files containing element names equivalent to each local element name.

The QIP design allows a user to use a consistent set of user-specified element names when formulating a query. These names are then translated by QIP into element names known at the host. The translation process is transparent to the user, in the sense that the user will not be aware of the host element name.

Data values within a query may have different formats or code conventions at different data bases. Data value transparency allows the user to specify a query using a consistent set of values, provided a value translation rule has been specified.

d. The Effects of Incomplete Standardization

The essential effect of incomplete standardization is that the user/analyst would interact with the network of data bases in a direct-to-host mode similar to the present practices. The TIIN user would have some productivity advantages over the present system, due to the transparencies provided by the TIIN System. The productivity advantages would include:

- One query language for all hosts
- One communications discipline for all hosts
- Standardized error messages
- One report formatting language for all hosts, with a standard default format
- Partial standardization of element names
- Partial standardization of data values.

However, the TIIN user would still be confronted with the problems of non-standardized element names and non-standardized data codes. That is, to access a host file the user/analyst would have to be thoroughly familiar with the element names and code conventions. This is not a problem unless the user seeks to combine similar data from different files. When similar elements have different code conventions, it might be necessary to use different queries to access separate files. More importantly, it would be difficult to combine the responses having different code conventions.

4. DATA REQUEST INTERMEDIATE FORMAT

The DRIF is a data structure used to pass queries from the user query language processor (e.g., TAP) to the User QIP. A single DRIF is produced as a result of each user query.

The DRIF is designed to be general and capable of handling a superset of all features appearing in query languages being translated. To allow this, the DRIF design is open-ended and can easily assimilate additions. The verbs and operators in DRIF are system operators, rather than language specific operators, allowing the addition of new operators without any changes to the structure of DRIF.

a. The Structure of DRIF

The DRIF consists of two kinds of information. General information about the query, such as user identifier, query identifier, and other fixed information are stored in the fixed portion of DRIF. The query itself is stored in reverse Polish notation in the conditional table in the variable length portion of DRIF.

b. Sources of Data

The information contained in the DRIF is derived from two sources: the user/analyst and the TIIN system. System provided information includes the user node name, the originating terminal identifier, and the time the query originated. Analyst-provided information includes data selection criteria, a list of data names for which data values are wanted, maximum number of responses the analyst wants, the desired response format, and additional destinations for the responses.

In order to account for all queries, the TIIN system will assign a unique identification to each query. The identification will consist of the user node at which the query originated, the user ID, and the time the query was originated.

The analyst may also provide information to locate the data. The analyst is permitted to specify a host data base and, optionally, the name of a file located at that data base. When the name of the data base is not specified the TIIN system will try to satisfy the query from all the data bases on the network. (In the TIIN prototype only one data base will be selected.) When the data base name is specified and the file name is not, the TIIN system will try to satisfy the query from the data located at the specified data base.

c. Language Independence

The structure of the DRIF is independent of the language in which the query was originally stated. The syntax of the original language is translated into the DRIF syntax, and all function names are translated into standard TIIN function codes at the time the DRIF is generated. Consequently, in order to interpret a DRIF, TIIN does not need to know anything about the source language of the query.

d. Data Base Dependence

The only parts of the DRIF not in the standard TIIN system format are the element names and element values. The module generating the DRIF is familiar with the query language it is translating, but is not familiar with the elements, since data elements are not part of the language. Different nodes in the system may use the same query language, but different sets of elements. The QNF Generator, the module that transforms the DRIF into the QNF, does have access to element name and value translation tables in the NAD (Network Access Directory) and therefore is the logical module to do the name and value translation.

The element names appearing in the DRIF can be divided into three categories: user names, TIIN standard names, and host names. User names are names that are known at the local node and that can be translated into TIIN standard names by using translation tables in the NAD. Host names are names local to the host DBMS and therefore not subject to the translation process. Whenever an analyst specifies any host names in his query, he is required to also specify the host data base.

5. QUERY NORMAL FORMAT (QNF)

The QNF is a data structure used to transfer queries from the user node to the host node. Its compact structure makes it particularly well suited for sending between network nodes.

For the TIIN prototype, a QNF corresponds to a single user request, and one user request produces a single QNF. At later development stages QIP will be capable of creating several QNF's from a single user request or DRIF to access similar data in several different files or data bases.

a. Structure of QNF

The physical and logical structure of the QNF is the same as the structure of DRIF. The difference between the two data structures is in the way they are used and in their content.

b. The Content of QNF

All the rules applying to the content of the DRIF also apply to the QNF together with a few additional restrictions.

Since a QNF is used to transmit queries between nodes of the network, its content must include the names of the host node, data base, and file, whereas these items are optional in the DRIF.

The QNF is language and data base independent. All operators, element names, and data values contained in it are system standard operators, names, and values. The only exception to this rule are host element names and values in queries utilizing the direct host access feature.

6. QUERY TREE FORMAT (QTF)

QTF is another format for expressing standardized format queries. The QTF is equivalent to the QNF, i.e., a QNF can be uniquely transformed into a QTF, and a QTF can be uniquely transformed into a QNF.

The difference between the QNF and the QTF is in the Conditional Table. The Conditional Table of a QNF contains an expression in reverse Polish notation. In the QTF, the expression is transformed into an equivalent tree data structure. Other portions of the QNF remain unchanged.

There are several advantages to transforming a query into QTF. A QTF contains direct links between operators and their operands, making it simple to examine relationships between operators and operands. A QTF clearly displays the hierarchical structure of a query.

The other advantage of the QTF is the ease of manipulating a tree structure. Since a tree structure does not depend on being stored in a contiguous amount of storage, nodes can easily be altered, added, and deleted, and it is possible to change a portion of the tree by changing pointers rather than by copying the entire structure.

In the TIIN prototype, the tree structure will be used at the host node by the QTF Processor. At later stages of development the QTF may be used by the User QIP if query validation or manipulation becomes necessary at the user node.

7. NETWORK ACCESS DIRECTORY (NAD)

The NAD is a distributed data base containing information obtained through the standardization process. Any module in QIP requiring information to translate or locate data names or values will obtain that information from the NAD.

The NAD has been designed under the Transparent Intelligence Language Facility (TILF) contract. The TILF design includes the internal structure of the NAD as well as procedures for creating and updating the NAD.

The NAD is used both by the User QIP and the Host QIP. QIP modules are permitted to reference the NAD, but are not allowed to change it.

a. Distributed Design

NAD is a distributed data base since a part of NAD exists at each node of the network. Each instance of the NAD contains different data, with little overlap between the instances. The data in NAD are distributed in such a way that a module utilizing NAD at a specific node will obtain all of its information from the local NAD.

The above brings out an important design trade-off. Having all information about the entire network available at each node would enable QIP to validate queries more completely at the user node, but would make the NAD very large and inefficient. Small amounts of data at each node will make the NAD efficient, but will result in more queries being rejected during validation at the host node.

b. Look-Up Functions

Each local NAD will contain information enabling QIP to translate local element names and values from the user frame-of-reference to the network standard, and vice-versa. Additionally, given a standard element name, the NAD will have the capability to provide a complete list of files containing equivalents of the standard element name. QIP will obtain the above information from the NAD by using three sets of look-up functions: one set for element name translation, one set for value translation, and one set for file selection.

c. Transparent Host Access vs. Direct Host Access

The mechanism of transparent access of elements and values between a user node and a host DBMS depends on two links: each TIIN standard element name must be equated to some user element name, and each host element name must be equivalenced to some TIIN standard element name. This allows the user NAD to have no information about actual host element names, while at the same time the host NAD need have no information about user element names. However, if either link is absent then the user does not have transparent access to the host data element. This situation has three possible solutions in the TIIN system.

(1) Addition of a User Equivalence

If the TIIN standard element name has not been given a local user element name, the user may request the local node administrator to update the NAD at the user node so that the TIIN standard element will be equated to a user specified element name. Subsequently the user will have transparent access to all host elements which have been equated to the TIIN standard element.

(2) Addition of a Host Equivalence

If a host element has not been equated to any TIIN standard element name then no users of the TIIN system have transparent access to the host element. This situation might occur for a variety of reasons such as security, infrequent use, data reliability, or complexity of standardization. Data access (and thus data analysis) for all users of the TIIN system is affected when a host element is equated to a TIIN standard element name. The responsibility for equivalences rests with both the host data base administrator and the TIIN network administrator.

(3) Direct Host Access

The analyst has the option of selectively or totally bypassing the transparent access mechanism for element names and/or data values. This option is available if the analyst is aware of elements which exist at the host DBMS but are not accessible via the host NAD. This option would be used extensively during the transition period while file standardization occurs on a step-by-step basis. Because transparent access is on an element basis, it will be possible to access the standardized element on a transparent basis, while the non-standardized elements would be accessed on a direct basis. The TIIN system also separates the issues of name standardization and value standardization. It is possible for the user/analyst to reference the element name on a transparent basis while the value encodement conventions for the element must be handled on a direct host basis.

d. Administration of the NAD

One of the basic assumptions of the TIIN design is that all elements defined in the NAD are available to all users of TIIN at all nodes. A data base administrator responsible for a single data base cannot choose to withhold information from selected nodes or analysts, but he does have the option of withholding information from the entire TIIN network by not allowing that information to be entered into the NAD.

The content of the NAD is controlled by the TIIN network administrator, who has the capability to create, delete, and change all local NAD's. It is the responsibility of the network administrator to define system standard elements and values, to obtain information about changes to data bases in the network from the individual data base administrators, to enter these changes into the NAD, and to ascertain that the NAD remains consistent at all times.

e. Future Extensions

The NAD described here contains a minimal set of functions and information necessary to a basic TIIN capability. Other kinds of data have been considered for inclusion in the NAD, but temporarily rejected. Some of this additional information may be added to the NAD at a later design stage to allow for the recognition of invalid queries at the user node, rather than at the host node.

In addition to the above functions, the host NAD will provide QIP with file structure information. The format of the structure information depends on the host DBMS, and is therefore different at each host node. At a DIAOLS data base, for example, the NAD will indicate which elements are inverted, while at a TILE data base the NAD will have to provide format numbers for all elements in multi-formatted files.

All host NAD's will have the capability to provide QIP with information on implicit elements. Implicit elements will not be translated into host elements, but instead NAD will contain information on implicit values of such elements.

8. USER QIP - QNF GENERATOR

The user QIP is that part of the QIP system which resides at the user node and operates on queries prior to dispatching them to the host node. The user QIP consists of a single module, the Query Normal Format (QNF) Generator.

The QNF Generator accepts a query generated by TAP and attempts to convert it into a QNF. If the conversion is successful, a QNF is generated and passed to QDNC. If any errors are detected, an error message is returned to the user and no QNF is generated.

In the TIIN prototype a DRIF will be translated into a single QNF (Query Normal Format) and therefore only a single file in the entire network will be queried as a result of a DRIF. At later stages, TIIN will have the capability to generate several QNF's from one DRIF to query multiple files in multiple data bases. When several files are queried as a result of one DRIF, all responses will be merged and presented to the user as a single report so that the entire process will be transparent to the user.

The QNF Generator at this stage has three main functions: element name translation, data value translation, and host selection. At later stages, the QNF Generator will include more sophisticated host selection algorithms as well as additional functions. One such function will be the creation of a file containing query status information.

a. Element Name Translation

(1) Local Names

Element names contained in the DRIF will usually be names local to the user node. Before transferring the query to a host data base, these element names must be

translated into TIIN standard element names. The NAD (Network Access Directory) at each data base has a table to provide for the translation of local element names at each node into TIIN standard names.

It is assumed that no two elements in one file may have the same meaning, and an element name always has a unique meaning within a data base. Consequently, each system element may correspond to at most one element name in each file within each data base in the network, and each element name available at a local node corresponds to at most one system element name. As a result, the look-up functions shown below always produce unique, possibly null, results:

$$SE = F_1 (LE)$$
$$LE = F_2 (SE, LF)$$

where

F_1 - local to standard element name translation function
F_2 - standard to local element name translation function
LE - local element name in the specified file
SE - network standard element name
LF - local file name

If a translate function produces a null result, i.e., an attempt is made to translate an element not in the NAD, the entire query will be declared invalid and the analyst notified.

(2) Direct Access to Host Element Names

An exception to the translation algorithm described above is the mechanism of direct host access. In certain cases, a user may know that an element exists at a host data base, but does not exist in his local language, and therefore is inaccessible by using the translation technique. In those cases, the user may enter the actual host element name in the query. The TAP language will have special notations to indicate which elements are direct-to-host. Such a name will be assumed to exist at the host data base specified by the user, and the QNF Generator will transfer it from DRIF to the QNF without attempting to translate it.

b. Data Value Translation

The problems associated with element name translation also apply to data value translation. The QNF Generator will use the following set of data value translation functions available from the NAD in translating local values included in a query into system standard values.

$$SV = G_1\ (LV,\ LE)$$
$$LV = G_2\ (SV,\ SE,\ LF)$$

where

G_1 - local to standard data value translation function
G_2 - standard to local data value translation function
LV - local data value for the specified element in the specified file
SV - network standard data value for the specified element
LE - local element name in the specified file
SE - network standard element name
LF - local file name

c. Host/File Selection

The host selection issue deals with the problem of deciding which data base and file contains the data required to respond to a query.

(1) Automatic File Selection

For the TIIN Prototype the QNF Generator will be restricted to choosing a single host file, even though more than one file may qualify. If more than one file is found to qualify for the query, a list of all qualifying files will be returned to the user. The user will be asked to select a file and resubmit the query. If no files in the network qualify, an error message will be returned. At a later stage, the QNF Generator will have the capability to query multiple files.

The NAD will provide the QNF Generator with information necessary to select host files. This information will consist of a list of data bases and files which contain a given element. This NAD function does not differentiate between implicit and explicit elements and consequently the list provided includes both types. The QNF Generator will be able to interrogate the NAD with the following function:

DB/F List = H (SE)

where

DB/F List	-	List of data base and file name pairs containing the desired element
H	-	host/file selector function
SE	-	network standard element name

The file selection algorithm will use the above information to select a host data base and file containing all the elements in the query by checking all the elements in the conditional table.

(2) Direct Host Access

The user has an option to specify a host data base, or a host data base and file. This feature is useful whenever a user wants to query a specific data base or file, rather than allowing the QNF Generator to select a host for the query. Whenever a DRIF contains a data base name, the QNF Generator will attempt to locate a single file within that data base to satisfy the query.

9. HOST QIP

The Host QIP is a set of translator modules located at the host node of TIIN. Their function is the translation of a QNF query into the query language used by the host DBMS.

An effort has been made to design a Host QIP that is table driven and query language independent. Due to the complexity and special requirements imposed by query languages, it was impossible to achieve these goals in full. Instead, the processor will consist of a number of independent modules, some of which are totally language independent, some table driven, and some that will have to be customized for each query language implemented.

In the course of translating a query, the Host QIP will convert the query into a Query Tree Format (QTF), validate it, perform language dependent transformations, and convert it into the host query language.

The Host QIP consists of four processes performing the above functions. The QTF Generator converts the QNF into the QTF. The QTF Processor validates and re-structures the QTF. The HL Generator converts the QTF into the HL. The HL Processor converts the generated HL string of characters into a finished query in the host language.

The following sections describe those host QIP modules that are applicable to most query languages. The language independent modules generally require a parameter or a table format description of a certain aspect of the query language. Language dependent modules are only described in general terms, since their specific function depends on the query language.

a. Query Tree Format (QTF)

This process consists of a single module which converts the reverse Polish notation expression in the QNF into the QTF. This module is completely language independent, does not require any tables, and can be used in any translator without any changes.

b. Query Tree Format (QTF) Processor

The QTF Processor has a number of separate functions and therefore is defined in terms of a series of modules, all of which operate on the QTF. The functions of the QTF Processor include element name and data value translation, syntax validation, and query optimization with respect to the host query language.

The modules described below are functional, rather than program modules. At implementation time, the functions may be partitioned differently when forming program modules. Most resulting program modules will operate on the QTF. Since the input and output data structures are the same for most of the modules, the executioon sequence of the modules can be changed even after implementation, including the capability to execute a single module several times, and to create and insert new modules into the sequence.

(1) Implicit Element Processing Module

Prequalified files can effectively be considered to contain additional, implicit, elements with fixed values in every record. As an example, a file describing the army troop strength in the United States contains two implicit elements, "service-branch" and "country", whose implicit values always are "army" and "United States." At the user node, implicit elements are treated in the same manner as explicit elements. At the host node the Implicit Element Processing Module identifies implicit elements, tests their values, and reduces the query accordingly. The resulting QTF does not contain any implicit elements, but is altered in such a way that it will extract from the data base only those records that were requested by the original query.

(2) Element Name Translation and Validation Module

Network standard element names are translated into host element names by using element translation functions available in the NAD. Direct-to-host names, that is names local to the host data base and placed into the QNF at the user node, are not translated or validated at this point, and are validated by the host DBMS. The validation function is fully table driven.

(3) Value Translation and Validation Module

This module translates element values in the QTF into equivalent values understood by the host data base. During the translation process all values that do not correspond to any host values will be rejected as invalid.

The value translation problem is extremely complex and will not be analyzed in detail within the scope of the TIIN/QIP effort. Translation of values, however, is an intrinsic part of the query translation process and will be studied under the proposed TIIN Response Normalization (RN) effort.

(4) Operator Existence Test Module

The purpose of this module is to check whether TIIN standard operators used in the QTF are all defined in the host query language. This module will not attempt to decide whether the query as a whole is valid in the host language. Instead, it will serve as a simple preliminary test designed to reject queries that are clearly incompatible.

The operator test module will be driven by a single, query language dependent table. The module itself will be language independent and will not require any modifications when implementing new host query languages. The table will contain a list of TIIN standard operators defined in the host query language. The program module will check each operator in the query against the list and reject the entire query is any operators in the query cannot be found in the operator list.

(5) Verb Selection Module

In the query languages reviewed, each query consists of a list of statements, each one starting with a verb.

Each kind of a verb contains different information about the request to the data base. It is not possible, however, to define a one-to-one correspondence between verbs in different languages because the statement definitions overlap. Consequently, information contained in a query is grouped into two statements in the QTF, the SELECT and PRINT statements. The SELECT statement describes all information regarding record selection criteria. The PRINT statement describes which fields of data are to be extracted from the data base. As the query arrives at the host data base, it is the job of the verb selection module to separate the query into several statements valid in the host language.

Since the verb selection process is highly language dependent and is not based on any standardized set of rules, each translator will require a custom designed verb selection module.

(6) Parentheses Depth Check Module

Even though a query in the QNF contains no explicit parentheses, parentheses may be implied by the precedence of the operators included in the query. A pair of parentheses is implied whenever the tree structure indicates that a lower precedence operator is to be evaluated before a higher precedence operator. An implied set of parentheses in the tree structure would be converted to an actual set of parentheses during host language (HL) query generation.

Since most query languages set a limit on the maximum nesting depth of parentheses, it is necessary to have a module which checks the parentheses in a query, and, if necessary and possible, transforms the query to eliminate some parentheses. The Parentheses Depth Check module will therefore accept a single parameter, the maximum nesting depth allowed in a language. By utilizing primitive tree operations, this module can do one of three things: accept the query as presented, reject the query, or accept the query after applying some tree transformations to make the query valid in the host query language.

Initially the transformations applied will be limited to factoring and distributing logical operators. If it is found at a later time that similar transformations can be performed on other types of operators, the Factor and Distribute routines will be generalized to accommodate the new operators. This module is query language independent and requires a single language dependent parameter, the nesting depth.

(7) Query Validity Test Module

In contrast to most other modules, the Query Validity Test module is highly query language dependent. Its function is to assure that the query specified in the QTF is compatible in all aspects with the host query language. The modules described above test the validity of those features that can be checked by a general algorithm applicable to several languages. As other such features are identified, other generalized modules will be added to the system. Every query language, however, also has unique restrictions not found in other languages. The query validity test is a module designed individually for each query language to test those features that have not been previously tested by other modules.

The QTF Processor will always assume that the QTF only contains functions valid according to the QTF specifications. This module, therefore, only checks for constructs valid in the QTF, but not in the host query language, since the host language is the more restrictive one.

(8) Optimization Module

In the QTF Processor, optimization is defined in the following way: whenever there exist two or more constructs expressing the same operation, and one of those constructs executes faster than the others, transforming the query into the most efficient construct is referred to as optimization.

Most query languages do provide the analyst with multiple constructs representing the same function. The following shows several examples of such constructs:

A BETWEEN B AND C	⟷	A GT B AND A LT C
A AND (B OR C)	⟷	A AND B OR A AND C
A OR B ALSO C	⟷	A AND C OR B AND C
A EQ (B, C, D)	⟷	A EQ B OR A EQ C OR A EQ D

The design of an optimization module for a language, therefore, consists of defining all sets of equivalent constructs in a language; for each such set, selecting the most efficient construct; and designing an optimization module which converts elegible constructs into their more efficient equivalents.

The optimization module cannot be designed as a table driven module. Instead, a superset of all possible optimizing transformations will be defined and programmed. The process of designing an optimization module for a language will then be reduced to selecting the appropriate optimization routines from the existing set.

c. Host Language (HL) Generator

It is the job of the HL Generator module to convert a query from the QTF into the host query language. It is assumed that a query is completely validated and optimized prior to executing this module, and therefore the HL Generator module does not need to perform any tests or transformations of the query.

The HL Generator is a table driven process, dependent on a single table and a driver program. The table contains a set of expansion rules, each rule describing the expansion of a single operator in the host language. The driver program is a general one, capable of interpreting a set of rules for any query language.

d. Host Language (HL) Processor

The HL Processor is a single-module, query language dependent process whose function is to convert the HL character string generated by the HL Generator into a finished query in the host language. The processing consists of checks that cannot easily be included in the HL Generator syntax driven format. The checks include line length checks, line number and end-of-line character insertion, and number-of-operators checks.

SECTION III

PRELIMINARY SPECIFICATIONS FOR DATA STRUCTURES

1. INTRODUCTION

This section contains the preliminary specifications for the following data structures:

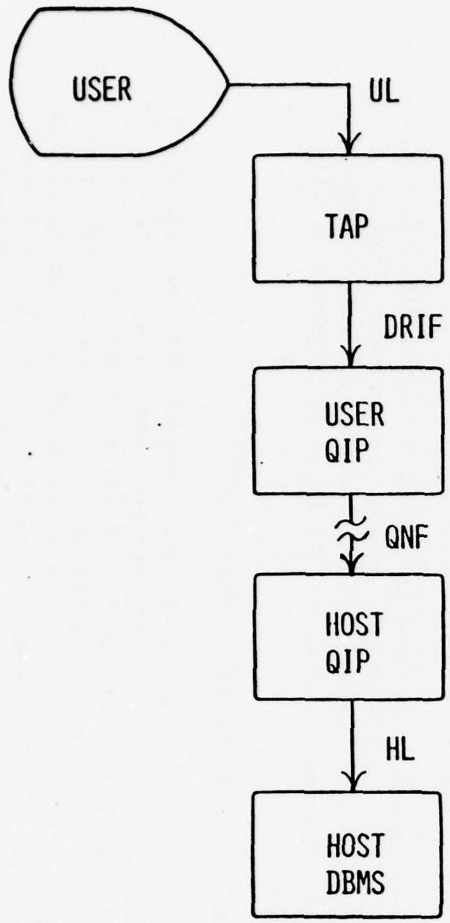
- o DRIF (Data Request Intermediate Format) is the data structure interface between the user language processor (e.g., TAP) and the User QIP.
- o QNF (Query Normal Format) is the data structure interface between the User QIP and the Host QIP.
- o QTF (Query Tree Format) is the data structure used in the Host QIP to perform algebraic transformations on the query.
- o The Syntax Table is the data structure used to convert the QTF version of a query to the Host Query Language version.

The user query language (UL) is the data structure interface between the actual user and the user language processor. Similarly, the host query language (HL) is the data structure interface between the Host QIP and the Host DBMS. The sequence of data structure interfaces, from UL to DRIF to QNF to QTF to HL, constitutes the user-to-host interface, and is summarized in Figure III-1. The host-to-user interface has a similar sequence and will be specified subsequently in the TIIN Response Normalization Study.

The data structures involved in the user-to-host interface are described in the following subsections, except for the user language (cf. Appendix H) and the host language (cf. Appendix A). Further presentation on how these structures are used can be found in Appendix B, TIIN Validation Exercises.

2. DATA REQUEST INTERMEDIATE FORMAT (DRIF)

The DRIF is the data structure which represents the user query for the interface between the user language processor (e.g., TAP) and the User QIP. There is one DRIF query generated for each user query. The functions and operators in the user query are represented by TIIN standard operator codes in the DRIF, and the infix notation of the user query has been converted to Reverse Polish Notation (RPN), so that the DRIF query is independent of the query language employed by the user to express the query. The DRIF is user-dependent in the sense that it may contain element names and element values based on the user's frame-of-reference rather than the host frame-of-reference or the TIIN frame-of-reference.



USER QUERY LANGUAGE

- o Infix Notation, English Keywords
- o User Element Names and Value Codes

DATA REQUEST INTERMEDIATE FORMAT

- o Reverse Polish Notation
- o TIIN Codes for Verbs and Operators
- o User Element Names and Value Codes
- o Optional Host

QUERY NORMAL FORMAT

- o Reverse Polish Notation
- o TIIN Codes for Verbs and Operators
- o TIIN Element Names and Value Codes
- o Specific Host

QUERY TREE FORMAT

- o Tree Notation
- o TIIN Codes for Verbs and Operators
- o Host Element Names and Value Codes

HOST QUERY LANGUAGE

- o Infix Notation, English Keywords
- o Host Element Names and Value Codes

Figure III-1. Summary of Characteristics of Data Structures Used in the User-to-Host Interface.

The layout and the codes for operators/verbs in the DRIF are the same as for the QNF and are described in the following subsection.

3. QUERY NORMAL FORMAT (QNF)

The QNF is the data structure which represents the user query for the interface between the User QIP and the Host QIP, as the query leaves the User node and as it is received at the Host node. (Under the TOSS operating environment there is another data structure interface, called WICS Common Format (WCF), in which the query is represented as a sequence of 256 word packets for transmission across the network to the actual host node. The conversion from QNF to WCF and back to QNF does not change the content or layout of the QNF and hence can be ignored in specifying the interfaces between the TIIN modules.)

If the user has explicitly specified the host file then there is one QNF query generated for each DRIF query. In the prototype TIIN the host file may be omitted by the user if it is uniquely implied by the element names in the query, so that there is one QNF query per DRIF query. In later development stages of TIIN it will be possible to query multiple files via one user query, so that one DRIF query may result in several differing QNF queries, addressing different host files. There is one response for each QNF query, and a composite response must be formed from the separate responses through merge, join, and difference operations corresponding to the way the separate QNF queries were formed from the original DRIF query.

Each QNF query has an explicitly designated host file as its destination. The user element names in the DRIF have been replaced by TIIN standard element names in the QNF. The user element values in the DRIF have been replaced by TIIN standard element values in the QNF. (The value conversion occurs only if a value conversion rule or mechanism has been specified for the associated user element. The details of the value conversion process are beyond the scope of this report and will be specified in the TIIN Response Normalization study.) The QNF is user-independent in the sense that all references to the user frame-of-reference have been replaced by equivalent names, values, verbs, and operations from the TIIN frame-of-reference.

The QNF can be divided into two sections: the header and the conditional table. The header contains information that the TIIN system uses in processing the user's data request. The conditional table contains the TIIN system translation of the user's data request.

The contents of the QNF are diagrammed in Figure III-2 and described below:

Word 0	USER NODE ID is the network identification of the node from which the query originated.
Word 1	ORIGINATING USER IDENTIFIER is the TIIN identification of the user/analyst who initiated the data request. This ID is unique within a node.
Words 2-3	TIME QUERY ORIGINATED is the date and time that the DRIF for the data request was constructed. Words 0-3 together constitute a unique identifier for the user request (DRIF).
Word 4	QNF IDENTIFIER is used to differentiate between the different QNF's originating from the same DRIF.
Words 5-6	FLAG WORDS are status words used to indicate the operational status of the message headed by this block.
Word 7	ORIGINATING TERMINAL IDENTIFIER is the TIIN identification of the terminal which initiated construction of the data request.
Word 8	SECURITY CLASSIFICATION is a one-word code indicating the classification of the user/analyst originating the data request.
Word 9	MAX NUMBER OF RESPONSES is the maximum number of records the user/analyst wants to retrieve.
Word 10	HOST DATA BASE IDENTIFIER is the network identification of the host DBMS which will process the user/analyst query. A blank value in this field in the DRIF received from the user language processor (TAP) will be replaced with a valid host identification by the host selector module.
Word 11	FORMAT CODE is the TIIN identification for the user specified response format.
Word 12	COUNT OF RETURNEES is the number of users to receive responses.
Words 13-(12+R)	RETURNEES is a list of TIIN one word user identifiers of users other than the original user who are to receive responses.

WORD		
0	USER NODE IDENTIFIER	
1	ORIGINATING USER IDENTIFIER	
2	TIME QUERY ORIGINATED	
3		
4	QNF IDENTIFIER	
5	FLAG WORDS	
6		
7	ORIGINATING TERMINAL IDENTIFIER	
8	SECURITY CLASSIFICATION	
9	MAX NUMBER OF RESPONSES	
10	HOST DATA BASE IDENTIFIER	
11	FORMAT CODE	
12	COUNT OF RETURNEES (=R)	
13	RETURNEES (1 WORD EACH) ⋮	
13+R	CONDITIONAL TABLE LENGTH (BYTE COUNT)	CONDITIONAL TABLE
14+R	CONDITIONAL TABLE ENTRY 1 ⋮ ⋮ CONDITIONAL TABLE ENTRY M	

Figure III-2. Layout of QNF and DRIF.

Word 13+R	CONDITIONAL TABLE LENGTH consists of two bytes and is a count of the number of bytes in the conditional table, including the two-byte length itself (where R=COUNT OF RETURNEES).
Words (14+R)-end	CONDITIONAL TABLE

The entries in the conditional table are variable length (in bytes). The entry format is shown in Figure III-3a. Each entry includes a byte count, a type code, and a string of bytes, as described in Figure III-3b. Figure III-3c lists the codes used in the ENTRY TYPE field of conditional table entries.

ENTRY BYTE COUNT	ENTRY TYPE
ENTRY STRING	

a. Layout for byte string entry in DRIF, QNF, and QTF.

ENTRY BYTE COUNT	a one-byte count of the bytes comprising the string entry (minimum 3 bytes) including the string field, the type field, and the byte-count field.
ENTRY TYPE	a one-byte code which describes the string field (cf. Figure III-3a).
ENTRY STRING	a byte-encoded operator or verb (cf. Figure III-5), or a string of characters for an element name, file name, or data value.

b. Specification of fields for string entries in DRIF, QNF and QTF.

1 = verb
2 = operator
3 = user element name
4 = TIIN standard element name
5 = host element name
6 = user value encodement
7 = TIIN standard value encodement
8 = host value encodement
9 = file name
10 = response operator
11 = full QNF identifier

c. Type Codes for byte string entries in DRIF, QNF, and QTF.

Figure III-3. Representation of Byte Strings in DRIF, QNF and QTF.

The conditional table is ordered so that the entries are in reverse Polish notation (RPN). The syntax description of the RPN is given in Figure III-4. The codes used in the ENTRY STRING field for operators are given in Figure III-5. Figure B-1 of the Validation Exercise shows an example of a user language query and Figure B-2 shows the corresponding DRIF Conditional Table.

The TIIN prototype will only be able to handle queries that can be expressed in terms of the entry types described here. The QNF provides the base representation of queries in a uniform network format. The QNF is designed to be flexible to allow for incorporation of features required for a specific user or host environment and features added in later stages of the TIIN Development. New Operators can be added by implementing new operator codes. The use of explicit codes to indicate whether names and values are in the user, the TIIN, or the host frame of reference provides flexibility to the user language processor.

a. Verbs

In the TIIN prototype the verb code differentiates between the select statement and the print statement with only one select and one print code per query. In later stages of development the verb codes may include data manipulation verbs, different kinds of retrieval and output verbs, as well as mixed sequences of verb phrases.

(1) SELECT (Verb Code: 1)

SELECT is a verb associated with the reverse Polish expression which qualifies the records to be extracted from the data base. The operators associated with the SELECT verb are described below.

(2) PRINT (Verb Code: 2)

The expression associated with PRINT specifies the elements to be extracted from the data base as a result of this query. In the TIIN prototype, the expression associated with PRINT is a list of element names, and cannot contain any operators.

b. Operators Used with the SELECT Verb

The operators defined for the TIIN prototype, and their associated TIIN standard operator codes are shown in Figure III-5. These codes are used in the DRIF, the QNF, and the QTF, and are used only with the SELECT verb.

```
<RPN-expression>      ::=   <multi-db-query>
                          | <complete-QNF-identifier> <multi-db-query>

<multi-db-query>      ::=   <single-db-query>
                          | <single-db-query> <multi-db-query>
                            <response-operator>

<single-db-query>     ::=   <phrase-list>
                          | <phrase-list> <file-list>

<file-list>           ::=   <file-specification>
                          | <file-specification> <file-list>

<response-operator>   ::=   MERGE | JOIN | DIFFERENCE

<phrase-list>         ::=   <verb-phrase>
                          | <phrase-list> <verb-phrase>

<verb-phrase>         ::=   <select-phrase>
                          | <print-phrase>

<select-phrase>       ::=   <select-expr> SELECT

<select-expr>         ::=   <criteria>
                          | <select-expr> <unary-logical>
                          | <select-expr> <select-expr> <binary-logical>

<criteria>            ::=   <geographic>
                          | <relational-expr>

<geographic>          ::=   <circle>
                          | <route>
                          | <polygon>

<circle>              ::=    radius <coord> CIRCLE

<route>               ::=    coord-count half-width <node-list> ROUTE

<polygon>             ::=    coord-count <node-list> POLYGON

<coord-list>          ::=   <coord>
                          | <coord> <coord-list>

<coord>               ::=    latitude  longitude

<relational-expr>     ::=   <expression> <expression> <relational-op>

<expression>          ::=    element-name
                          |  element-value
                          |  element-name <quantifier>

<relational-op>       ::=    GE | GT | LE | LT | EQ | NE | CONTAINS

<quantifier>          ::=    ALL

<binary-logical>      ::=    AND | OR | WITH

<unary-logical>       ::=    NOT

<print-phrase>        ::=   <element-list> PRINT

<element-list>        ::=    element-name
                          |  element-name <element-list>
```

Figure III-4. Syntax Description of the Reverse Polish Notation (RPN) Used in the Conditional Table of the QNF.

Operator	TIIN Standard Code	Number of Operands
Logical		
AND	1	2
OR	2	2
NOT	3	1
WITH	4	2
Relational		
CONTAINS (has)	10	2
GT (greater than)	11	2
GE (greater than or equal)	12	2
LT (less than)	13	2
LE (less than or equal)	14	2
EQ (equal)	15	2
NE (not equal)	16	2
Geographic		
CIRCLE	20	3
ROUTE	21	variable (min. 6)
POLYGON	22	variable (min. 7)
Quantifiers		
ALL	30	1
Response		
MERGE	40	2
JOIN	41	2
DIFFERENCE	42	2

Figure III-5. Standard Operator Codes in DRIF, QNF, and QTF.

(1) Logical Operators

The logical operators AND, OR and WITH must each have two operands. The logical operator NOT must have one operand. All operands associated with logical operators must generate Boolean values when evaluated. An operand of a logical operator must therefore be an expression formed from either another logical operator, a relational operator, a geographic operator, or any other operator resulting in a Boolean value.

The AND and WITH operators have the same meaning unless a hierarchy of record segments is involved. In a hierarchy where several subordinate segments have the same parent segment, the AND operator indicates the criteria may be satisfied in different subordinate segments under the common parent, whereas the WITH operator indicates the criteria must be satisfied in the same subordinate segment.

(2) Relational Operators

Seven relational operators (GT, GE, LT, LE, EQ, NE, CONTAINS) can be specified in the QNF. Each relational operator must have two operands. If one operand is an element name and the other one is a value or an expression, the element name must be specified as the first operand (i.e., 5 GT A would be converted to A LT 5). The first six operands specify the standard six relationships. The operator CONTAINS is used to select element values which include the specified partial value.

(3) CIRCLE

Operands: Radius, latitude, longitude

The CIRCLE operator tests whether the unit described in the current record in the data base is inside or outside of a specified circle, and returns a Boolean result. The operator must always have three operands. The first operand specifies the radius of the circle in nautical miles. The second and third arguments specify the latitude and longitude of the center of the circle. The latitude and longitude are specified in the following format: DDDMMSSg, where DDD are degrees, MM are minutes, SS are seconds, and g specifies E, W, N or S. All units inside the circle will be retained, unless the function is qualified by a NOT operator in order to retain all units outside the circle.

(4) ROUTE

Operands: Number of nodes, half-width, latitude, longitude, ...

The ROUTE operator tests whether a given location is within a specified route. A route is defined by specifying the half-width distance and the coordinates of each node defining the route. The ROUTE operator contains six operands for the simplest route consisting of two nodes, and another two operands for each additional node within the route. The first operand indicates the number of points in the specified route. This number must be greater or equal to two. The second operand contains a number equal to the half-width of the route, in nautical miles. The remaining operands are pairs of numbers, the first number of each pair specifying the latitude, and the second number specifying the longitude of a node. The number of such pairs must be equal to the number of nodes specified in the first operand. All units inside the route will be retained, unless the function is qualified by a NOT operator in order to retain all units outside the route.

(5) POLYGON

Operands: Number of nodes, latitude, longitude . . .

The POLYGON operator tests whether a given set of coordinates specifies a point inside or outside a polygon. The polygon is defined by providing a list of at least three coordinate points. The points must be listed consecutively clockwise or counterclockwise. The polygon operator requires at least seven operands. The first operand specifies the total number of nodes comprising the polygon, at least three. The remaining operands are pairs of latitudes and longitudes specifying the nodes. The number of latitude and longitude pairs must equal the number of nodes specified in the first operand. All units inside the polygon will be retained, unless the function is qualified by a NOT operator in order to retain all units outside the polygon.

(6) Quantifiers

Quantifiers are a general class of operators applied to repeating elements. When referring to a repeating element name in a query, an analyst actually refers to all the values for that element in one record. Quantifiers allow a user to refer to specific value(s) within a record, such as FIRST, LAST, NO, ALL. The initial set of TIIN standard operators allows only ALL.

c. Response Operators

Response operators are binary functions specifying the rules for combining multiple responses to query.

Currently, three such operators are proposed, MERG, JOIN, and DIFFERENCE, corresponding roughly to the OR, AND, and AND NOT logical operators, respectively.

The MERG operator matches records from the two input files on a one-to-one basis, forming composite records. The records are matched by comparing the values of a specified set of elements appearing in both files. Records lacking a match in the other file are discarded.

The DIFFERENCE operator matches records from two files in the same manner as the JOIN operator. Only records existing in the first file, but having no corresponding matches in the second file, are kept.

4. QUERY TREE FORMAT (QTF)

The QTF is a data structure in which the set of operands for each operator are explicitly linked together through the use of pointers.

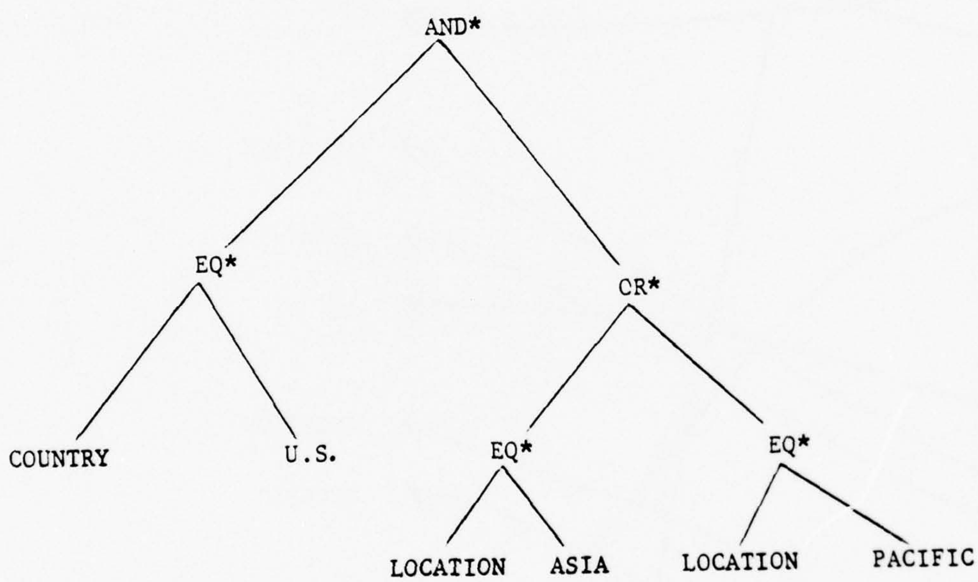
The symbolic representation of a QTF query consists of an upside-down tree with the "root" at the top and the "leaves" at the bottom, as illustrated in Figure III-6b. Each node or branch point has an operator, one or more "sons" (operands), and at most one "father" node (of which it is an operand).

The internal representation of a QTF query consists of the QNF version (which represents the "leaves") and a collection of "tuples", as illustrated in Figure III-6c. Each tuple occupies N+2 words of computer memory, where N is the number of operands or sons. Each tuple represents one node of the tree structure, and directly includes the operator, the operands, and a count of the operands. This information is encoded as follows:

Word 1 - operand count (=N)
Word 2 - operator pointer
Words 3 through (N+2)-operand pointers

```
     COUNTRY = 'U.S.'
AND (LOCATION = 'ASIA' OR
     LOCATION = 'PACIFIC')
```

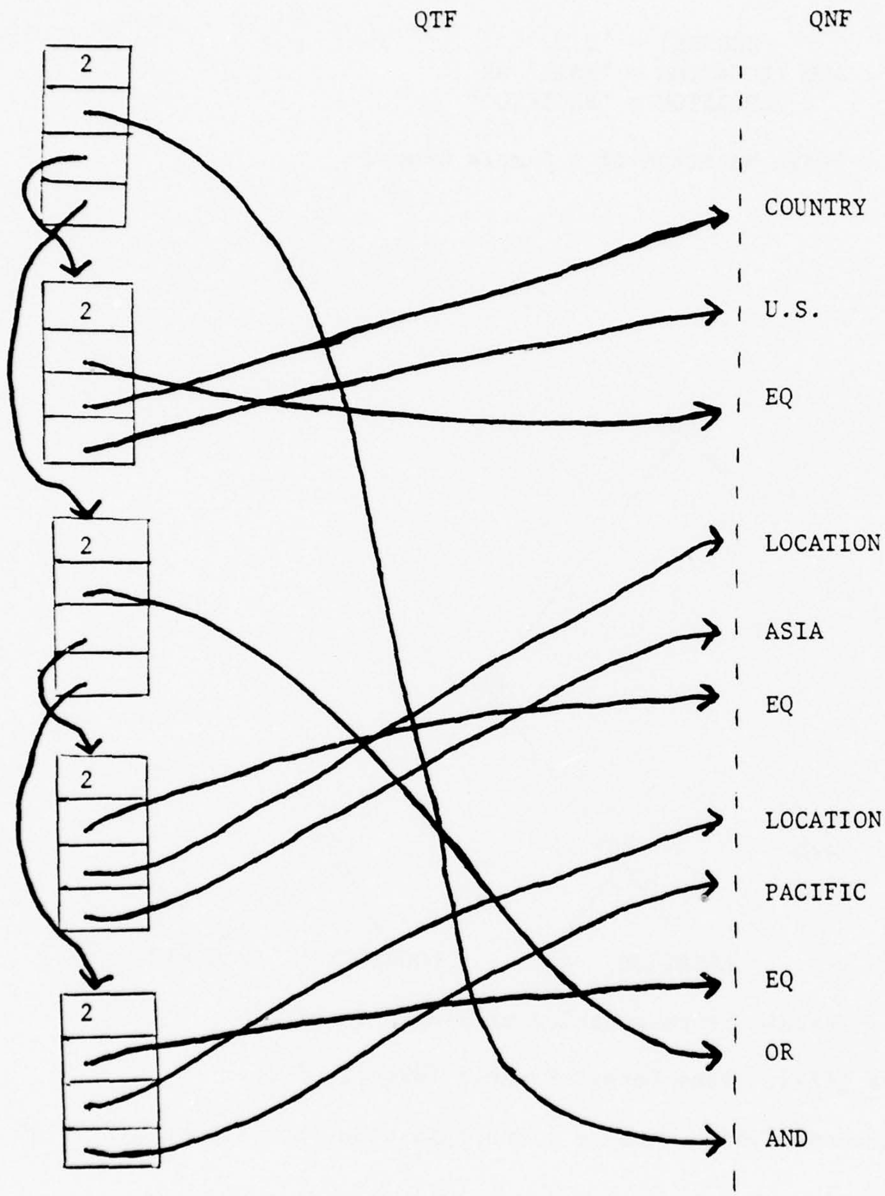
a. Infix Notation of a Sample Query.



b. Graphic Representation of Sample Query.

Figure III-6. Tree Format Example (Page 1 of 2).

*Indicates mnemonic which denotes a one-byte code from Figure III-5.



c. Internal Computer Representation of Sample Query.

Figure III-6. Tree Format Example (Page 2 of 2).

The operand count in word 1 contains the number of operands specified in the tuple (N), and therefore also determines the size of the tuple, which is always equal to N+2.

The operator pointer in word 2 points to a QNF entry containing the operator code, or to a similarly formatted entry outside of the QNF. The pointer is always specified as an offset from the beginning of the buffer containing the QNF.

The remaining words of the tuple, words 3 through N+2, contain pointers to operands. The total number of operand pointers is equal to the count (N) specified in word 1 of the tuple. Each operand pointer specifies the offset of the operand from the beginning of the buffer allocated to QNF or QTF. A pointer will point to an entry in the QNF if the operand is an element name or a value, otherwise it points to another tuple. The high-order bit of the 16-bit pointer is used to differentiate between the two cases, thus allowing a maximum of 32K for the QNF or the QTF buffer. Bit 0 of the pointer set to 0 indicates a pointer to an QNF entry, and bit 0 set to 1 indicates a pointer to another tuple.

There is an additional restriction on the data structure. In order to assure the integrity of the tree structure subsequent to restructuring operations, each node is only allowed to have a single father, i.e., no two pointers are permitted to point to the same tuple. It is the responsibility of the QTF Generator and Processor to enforce this rule.

5. STRUCTURE OF THE SYNTAX TABLE

The syntax table contains information about the host query language. The information is in the form of expansion rules which are instructions on how to expand the nodes of the QTF into a valid query.

Each expansion rule describes the expansion of a particular kind of a QTF node. The syntax table consists of two parts, a fixed format part and a variable format part. The fixed format part contains an entry for each type of node, each entry consisting of the node identifier, a precedence code, and a pointer to an expansion rule. The expansion rules describe how to convert the QTF node into the host language and are contained in the variable format part of the Syntax Table. Figure III-7 shows the symbolic format of the Syntax table.

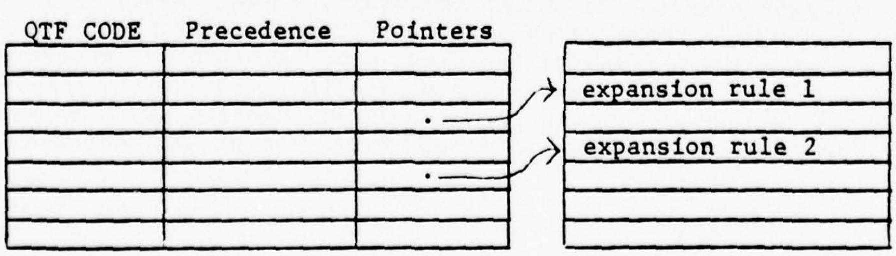


Figure III-7. Internal Representation of Syntax Table.

Expansion rules consist of three components: character strings, routine calls, and end-of-rule symbols. The components are interpreted in the sequence found in the rule, and the resulting host language character string is accumulated in the output buffer. This process is described in detail in Section IV-5.

Character strings included in the expansion rule are transferred unchanged into the output buffer. Routine calls process the descendants (operands) of a node and result in a host language expansion of a node and all of its descendants. The end-of-rule symbol marks the end of an expansion rule.

The following routine calls have been defined for inclusion in expansion rules:

CNVn - converts a single node into a host language character string. "n" may take on values 0, 1, 2, 3, etc. "n=0" indicates the current node is to be converted. All other values of n indicate the n'th descendant of the current node is to be converted.

CNVNn - converts a variable number of nodes into a host language character string. All descendants of current node, starting with the n'th descendant and ending with the last one, are converted. The character string in the expansion rule following this routine call is inserted between each pair of nodes converted.

Figure IV-24 contains the Syntax table for the DIAOLS language. The following describes the interpretation of some of the rules in DIAOLS.

The VALUE operand node is expanded into a character string (containing a colon) followed by the contents of the node (a value) followed by another character string (a colon).

The AND operator node is converted into the expansion of the first descendant (which may be a relational expression or a complex Boolean expression) followed by the character string '␣AND␣' followed by the expansion of the second descendant.

The AREA operator in DIAOLS contains the description of a geographic area and is interpreted as follows: output the expansion of the first descendant (the name of the area, such as CIRCLE-3) followed by the characters '␣EQ␣' followed by the expansion of the second descendant, a space, third descendant, space, etc., until all descendants have been processed. The last item generated is the expansion of the last descendant.

SECTION IV

PRELIMINARY SPECIFICATION OF QIP MODULES

1. INTRODUCTION

Section II of this report describes the component modules of QIP and the functions performed by each module. Section III describes in detail the data structures utilized by the modules. The purpose of this section is to present how the functions described earlier are performed. The explanations included here assume that the reader is familiar with the previous sections of this document.

Subsections 2 through 6 of this section describe in detail the language-independent parts of QIP. Subsection 7 describes those QIP algorithms that are unique to the DIAOLS query language.

2. QNF GENERATOR

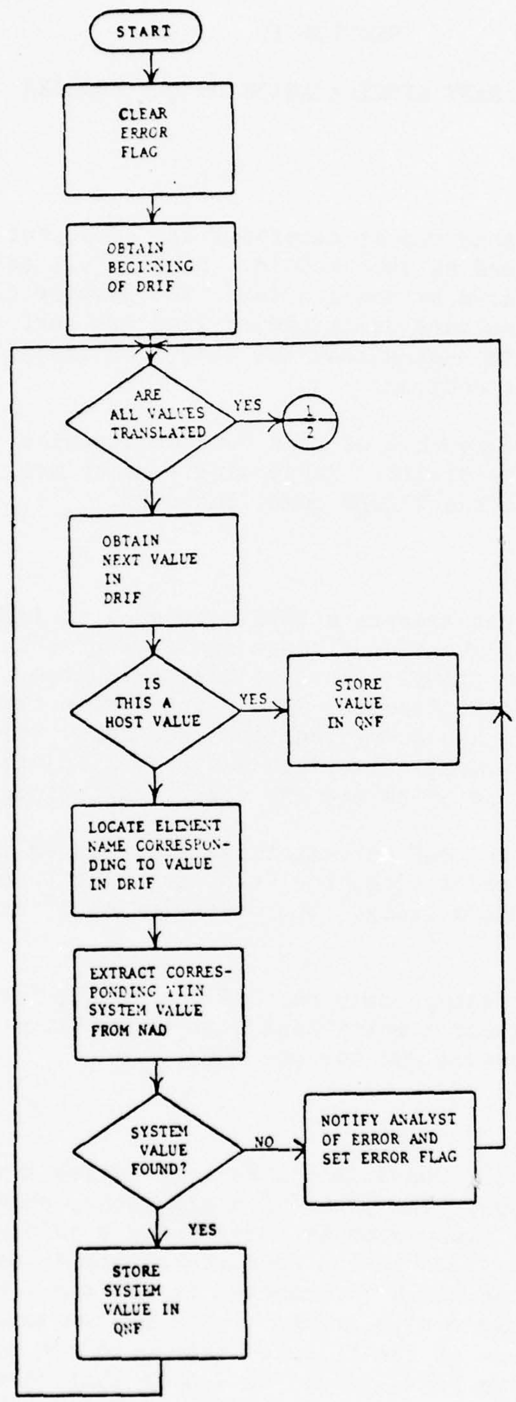
The QNF Generator accepts a DRIF, converts it into a QNF, and passes it to QDNC. The conversion process consists of three steps: value translation, element name translation, and host selection. All three steps utilize information obtained from the NAD. Since value translation is element relative, and the NAD query function requires a local element name and value to translate a local value into a system standard value, value translation is performed prior to element name translation.

At each step, the QNF Generator can find errors in the query and reject it. By using an error flag, the Generator will locate all errors in a DRIF before rejecting a query. The logical flow of the process is shown in Figure IV-1.

In the TIIN prototype only one QNF can result from one DRIF. In later development stages for a multi-host query capability it will be necessary to generate several QNF for one DRIF.

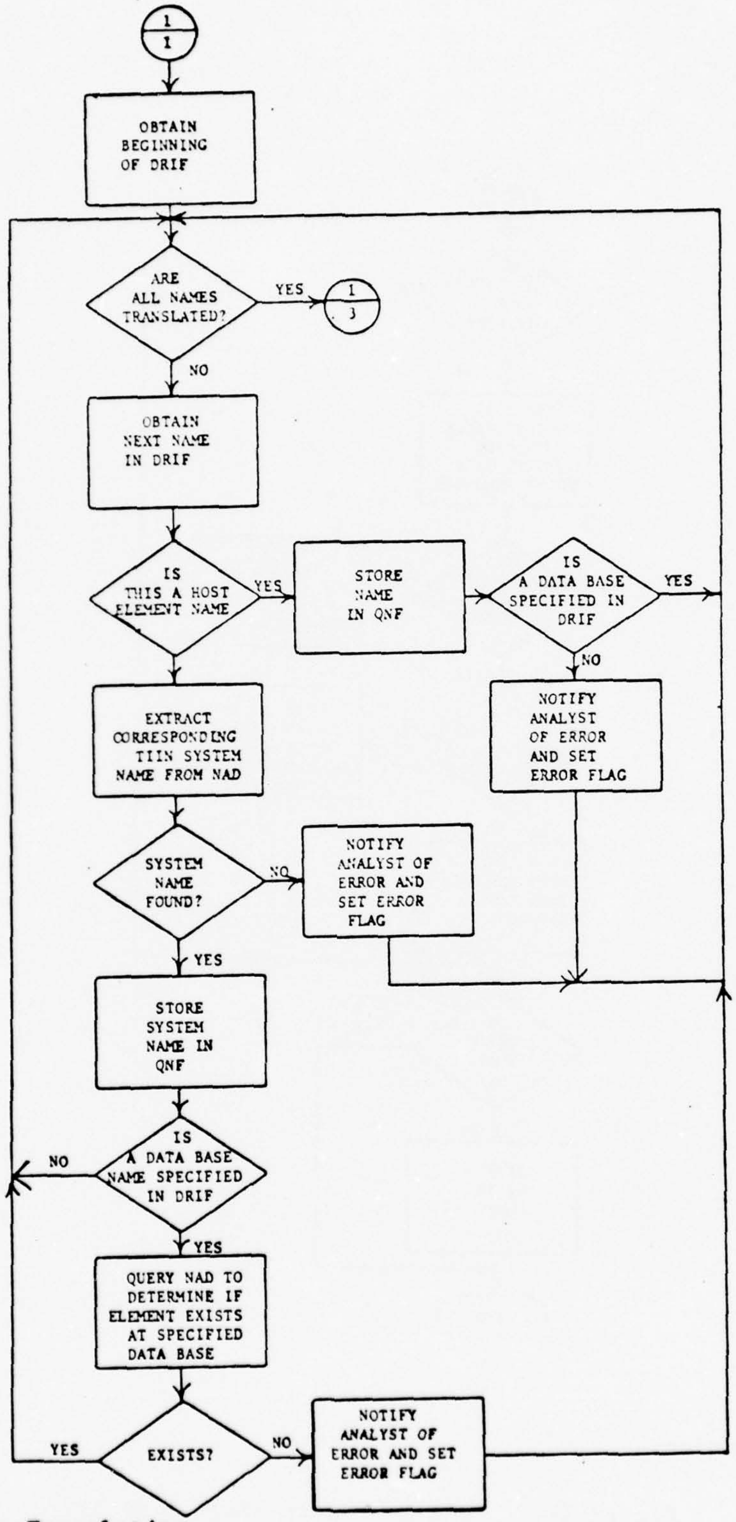
3. QTF GENERATOR

The QTF Generator converts the Reverse Polish expression in a QNF into a tree data structure. The conversion algorithm, shown in Figure IV-2, reads the Reverse Polish expression by maintaining a pointer which is advanced to point at successive entries in the Conditional table; uses a stack as intermediate storage of pointers to branches of the tree; has the capability to obtain storage and create tree nodes; and completes execution by pushing a pointer to the root node of the finished tree onto the empty stack. Not shown in the flowchart are tests needed to assure that no elements are popped from a functionally empty stack and that the stack has been returned to its initial state at the end of execution.



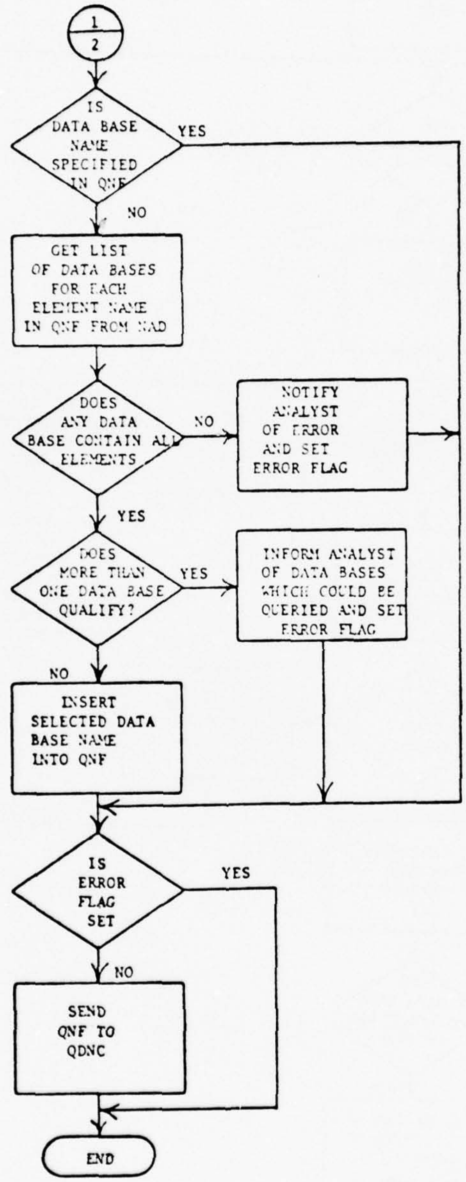
a. Data Value Translation.

Figure IV-1. QNF Generator (Page 1 of 3).



b. Element Name Translation.

Figure IV-1. QNF Generator (Page 2 of 3).



c. Host Selection.

Figure IV-1. QNF Generator (Page 3 of 3).

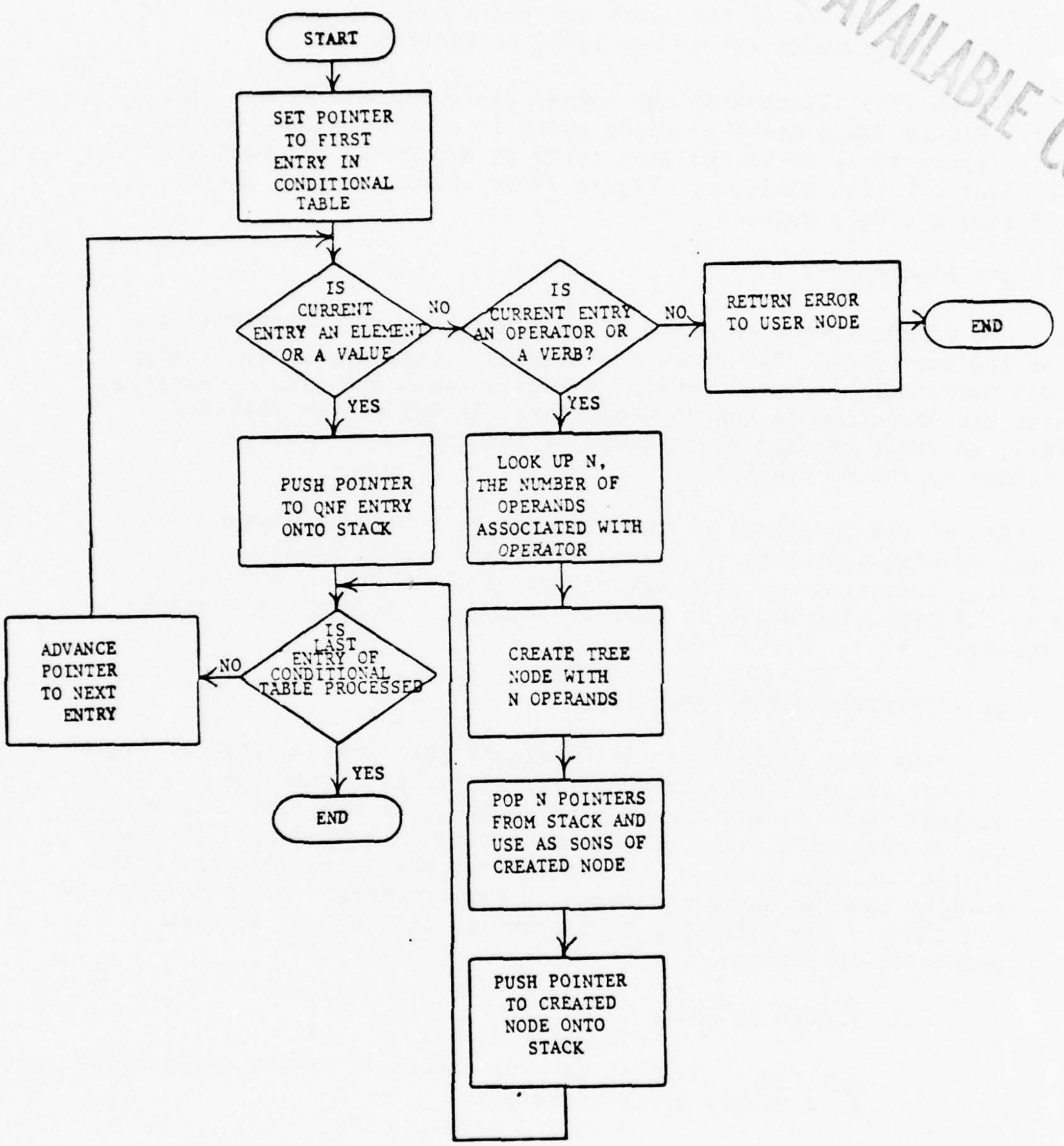


Figure IV-2. QTF Generator.

The root node is the last node resulting from the QTF Generation. The root node has an operator pointer of Ø, a file name as its first operand and the verb nodes of the query are the remaining operands. The operands of all other nodes are as described in Section III.

Figure IV-3 illustrates the conversion performed by the QTF Generator. Figure IV-3a shows a sample query in a hypothetical query language. Figure IV-3b shows the same query as stored in the Conditional table in Reverse Polish Notation. Figure IV-3c shows the same query converted into a tree structure.

4. QTF PROCESSOR

The QTF Processor consists of a number of tree operations performed at the host node. The Processor accepts a tree structure created by the QTF Generator, performs several validity tests and transformations, and passes the structure to the HL Generator. If any of the validity tests fail, an error message is returned to the user node and the query is not passed to the HL Generator.

All of the functions of the QTF Processor will be designed as independent modules operating on a tree structure. This design will allow for easy insertion and deletion of any of the modules, as well as changes in the execution sequence and, if necessary, multiple executions of the same module.

a. Primitive Tree Operations

The tree transformation functions described in the following subsections analyze a tree and transform it. While the logical flow of each function is unique, all functions include common primitive transformations. To avoid redundancy, the TIIN design includes a set of primitive tree operation routines which will be used as building blocks in constructing tree transformation functions. The following is a proposed initial set of such primitive functions.

(1) Binary to n-ary

Function: Convert several identical binary operations to a single n-ary operation.

When used: Used to redistribute a tree prior to verb selection in DIAOLS. This is necessary since different verbs can only be assigned to branches connected with AND operators.

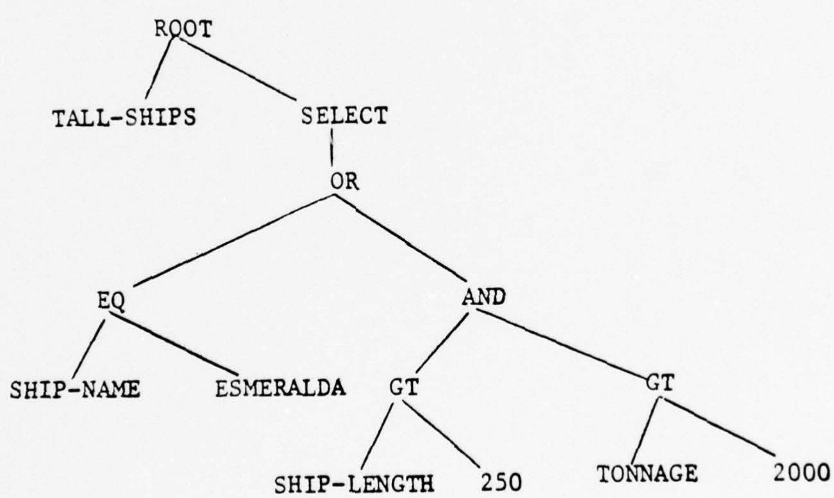
How it works: See Figure IV-4.

```
FILE TALL-SHIPS;
RETRIEVE SHIP-NAME = 'ESMERALDA' OR (SHIP-LENGTH > 250 AND TONNAGE > 2000);
```

a. Source Language Query.

```
SHIP-NAME
ESMERALDA
EQ
SHIP-LENGTH
250
GT
TONNAGE
2000
GT
AND
OR
SELECT
TALL-SHIPS
```

b. A Query in Reverse Polish Notation.



c. Same Query Converted into a Tree.

Figure IV-3. Sample Conversion from Reverse Polish Notation to Tree Format.

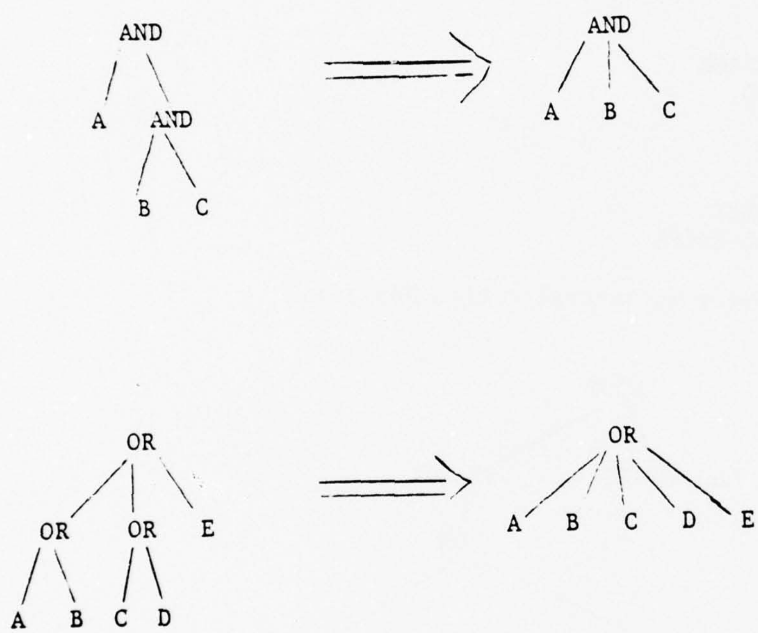


Figure IV-4. Examples of Binary to N-ary Transformations.

Arguments accepted: A pointer to a node, at least one of whose sons has the same operator as the node.

Arguments returned: None.

(2) N-ary to binary

Function: Convert an n-ary operation to a string of binary operations. This operation is the reverse of the Binary to n-ary operation.

When used: After verb selection in DIAOLS, used to convert all operators to binary format, as required by most query languages.

How it works: See Figure IV-5.

Arguments accepted: A pointer to a node containing an n-ary operator and a flag to indicate left or right recursion.

Arguments returned: None.

(3) Factor

Function: Factors a common sub-tree from all the sons of a node.

When used: Used to simplify expressions and to reorganize trees during verb selection in DIAOLS.

How it works: See Figure IV-6.

Arguments accepted: A pointer to the tree to be factored. The tree must have at least two branches, pointing to nodes with identical operators and with at least one common branch.

Arguments returned: None

Note: Not all operators can be factored without changing the meaning of an expression. The calling routine must decide if an operator can be factored.

(4) Distribute

Function: Distributes an expression over a tree. This is the reverse of a factoring operation.

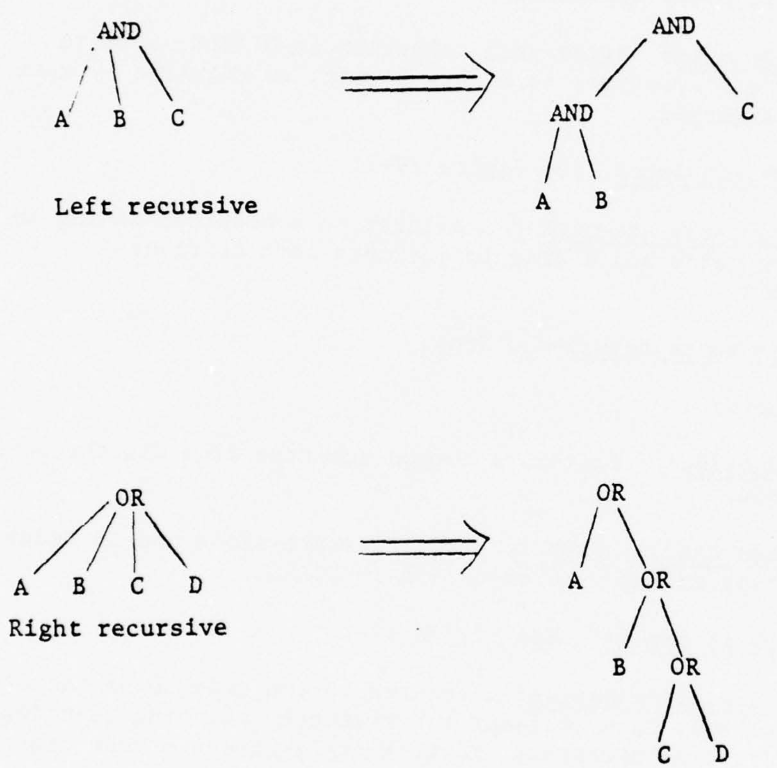


Figure IV-5. Examples of N-ary to Binary Transformations .

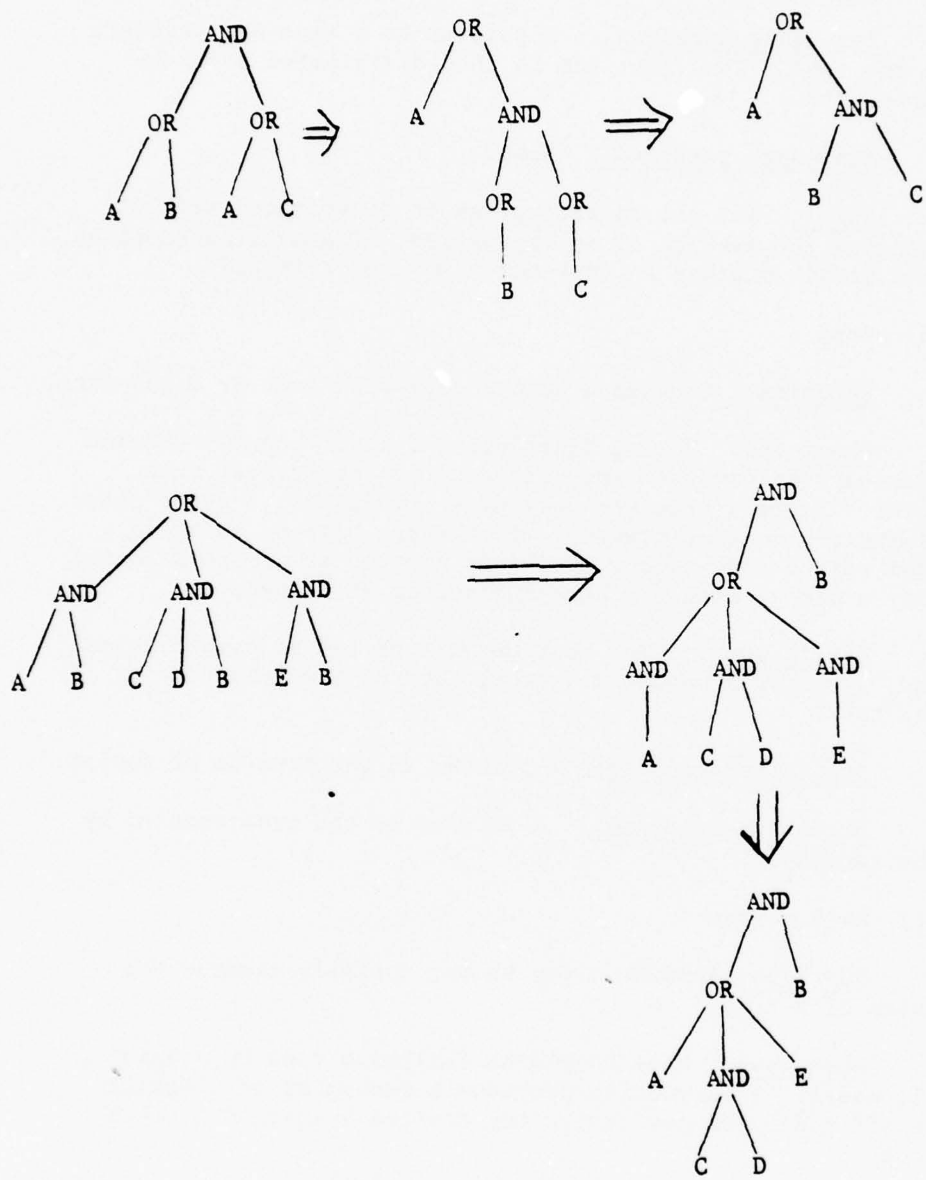


Figure IV-6. Examples of Factoring.

When used: To reduce the number of sets of parentheses in an expression, and in DIAOLS to reformat a tree prior to verb selection.

How it works: See Figure IV-7.

Arguments accepted: A pointer to a tree and pointers to two sons. The first son is then distributed over the second son.

Arguments returned: None.

Note: Not all operators can be distributed without changing the meaning of an expression. The calling routine must decide whether an operator can be distributed.

(5) *Copy*

Function: Creates a second copy of a tree or a sub-tree.

When used: During distribution, it frequently becomes necessary to use the same node as a son of several other nodes. Since a node may only have one father, a second copy of the son must be created. If the son happens to be an atom rather than a node, there is no need to create a second copy since an atom may have any number of fathers.

How it works: The routine obtains the necessary space and, by using a push-down stack, creates a second copy of the tree.

Arguments accepted: A pointer to the tree to be copied.

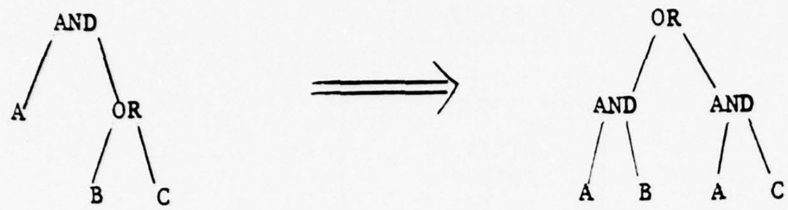
Arguments returned: A pointer to the tree created by the routine.

(6) *Walk a Tree*

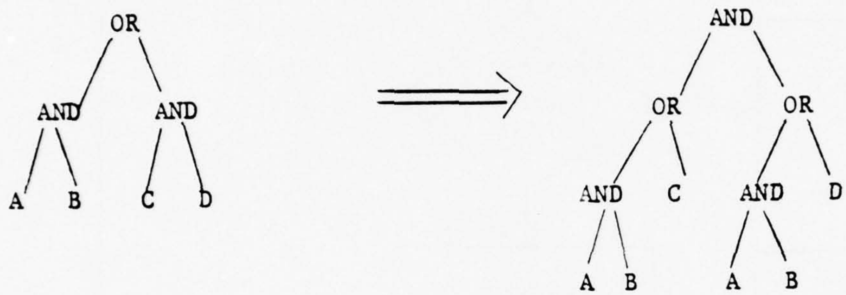
Function: Provide a way to sequentially examine all nodes of a tree.

When used: Many functions analyze a tree by examining all nodes. This routine provides a mechanism for looking at all nodes and performing any desired operation at each node.

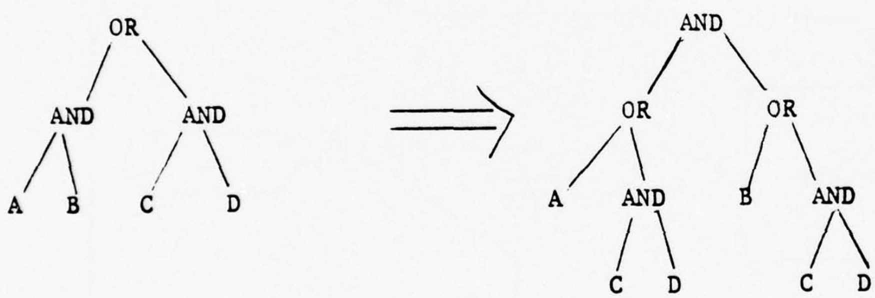
How it works: See algorithm in Figure IV-8.



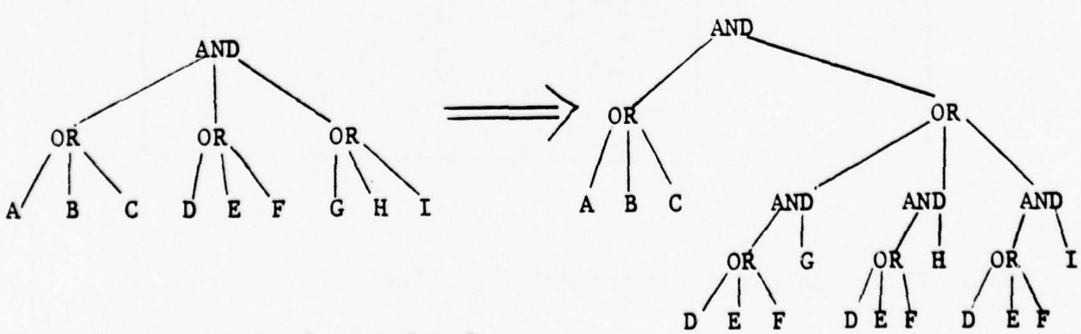
a. Distribute Son 1 over Son 2.



b. Distribute Son 1 over Son 2.



c. Distribute Son 2 over Son 1.



d. Distribute Son 2 over Son 3.

Figure IV-7. Examples of Distribution.

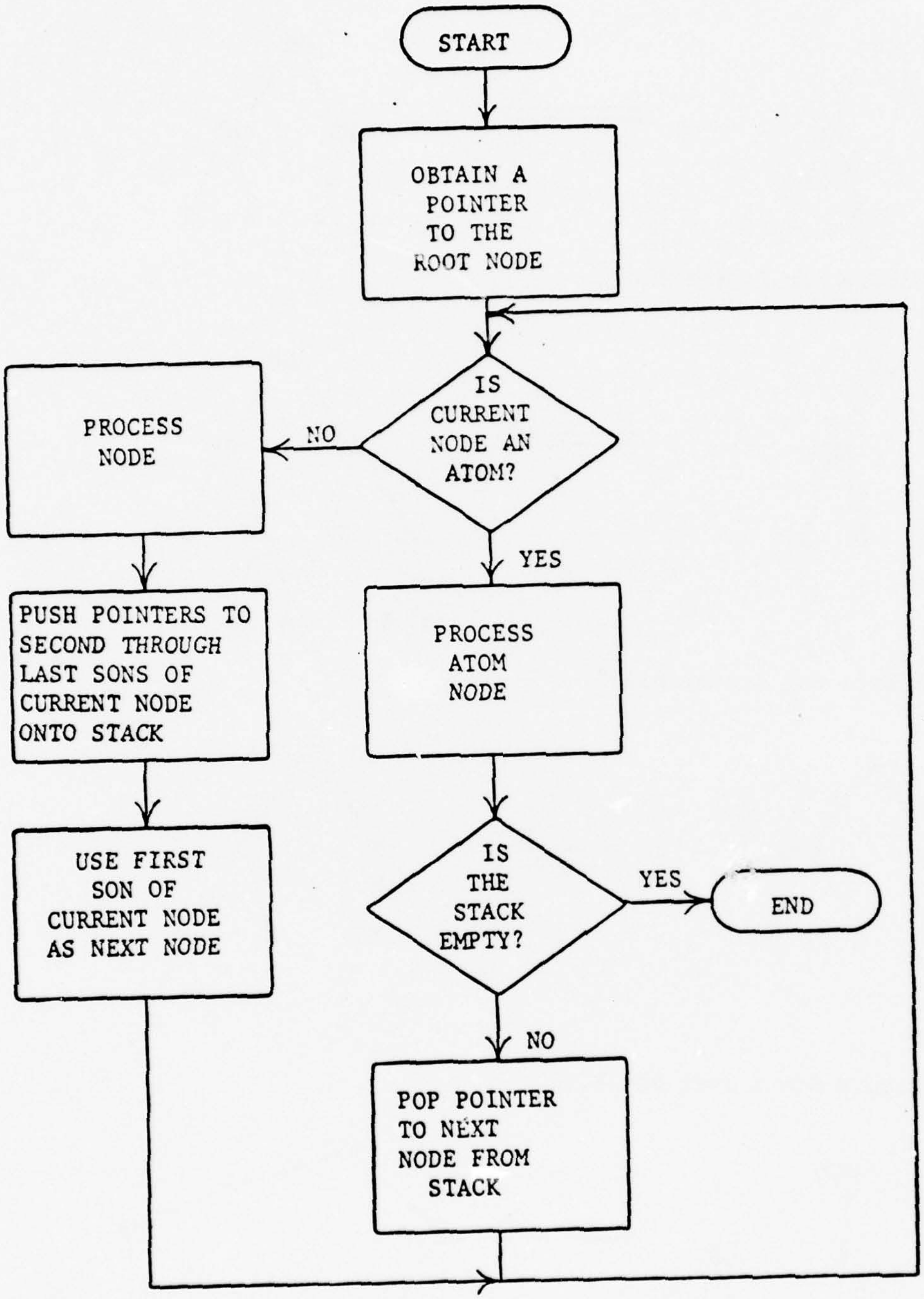


Figure IV-8. An Algorithm for Walking a Tree.

Arguments accepted: Names of routines to be executed at each atom and node.

Arguments returned: Error indicator.

Note: This routine controls the node execution sequence and initiates routines that process individual nodes.

b. Implicit Element Processing

The purpose of this module is to identify and eliminate implicit elements from the QTF. Implicit elements must be eliminated since, by definition, they do not explicity exist in a data base and therefore cannot be used when querying a data base.

The process described here consists of three distinct steps. The first step identifies relational expressions containing implicit elements and substitutes truth values in their place. The second step performs transformations to eliminate from the QTF the truth values inserted by the first step. The last step examines the remaining structure and determines if the QTF still contains a valid query.

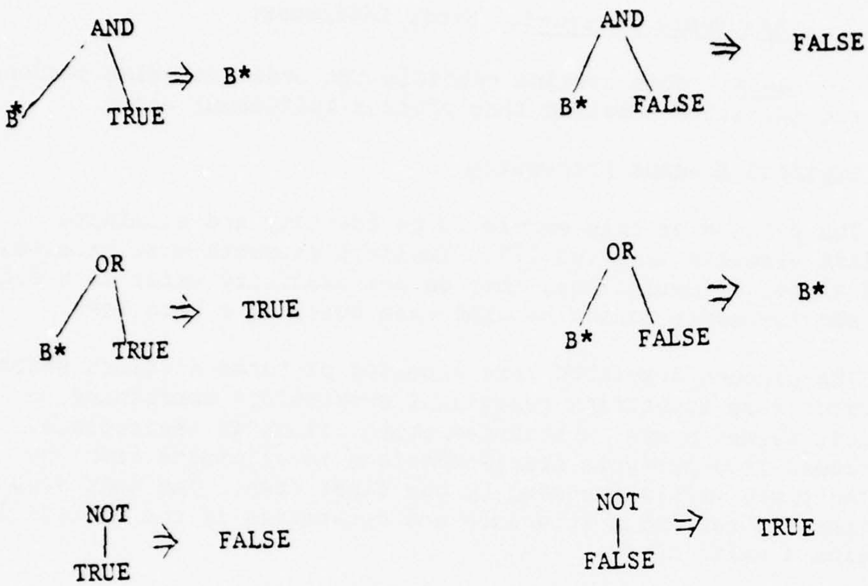
(1) Elimination of Implicit Elements

This process is performed by applying the tree walk primitive operation to locate tree nodes containing relational operators. Whenever a relational operator is found, the corresponding element name is located and submitted to the NAD as an argument of the NAD implicit element query function. The NAD response is either negative to indicate the element is not an implicit element, or positive to indicate the element is implicit. In the case of a positive response, the implied value of the element is also returned.

For each implicit element found, the implied value of the element is substituted into the relational expression in the QTF and the resulting expression (consisting of a relational operator and two values), is evaluated, producing a "true" or "false" value. The truth value is inserted into the QTF in place of the original relational expression.

(2) Query Reduction

The QTF at this stage contains Boolean operations on truth values. The results of such operations are independent of the content of the data base and can be evaluated at this stage by applying the transformations shown in Figure IV-9. The transformations are applied to the QTF until no more eligible constructs can be found.



*B denotes a subexpression of the Boolean expression

Figure IV-9. Boolean Value Elimination Transformations.

(3) Query Validity Test

The QTF which results from Boolean value elimination will be either "true", "false", or a Boolean expression of search criteria. In the first case, the query will accept every record on the host file. In the "false" case, there will be no data retrieved from this host file. In the third case, the reduced QTF has no implicit data elements and can be passed to the next processing step.

c. Data Value Translation and Validation

The purpose of this module is to translate system standard data values in a query into values local to the host data base.

The value translation and validation functions are performed by querying the NAD with the appropriate functions. The definition of the NAD query functions requires that value translation be performed prior to element name translation.

Each NAD request results in a unique answer or an error message indicating that the value does not exist in the specified data base and file. Whenever an invalid value is found an error message is sent to the user node. The processor at the user node converts the message to user format for delivery to the analyst.

Figure IV-10 shows the logical flow of this module. It is assumed that the processes are controlled by the tree walk module (Figure IV-8) and each time an atom unit is encountered, the translation module described here is invoked. The module processes atoms containing values, and ignores atoms containing element names. Whenever a data value is found, a corresponding element name is located to be used in the NAD query function.

d. Element Name Translation and Validation

The element name translation module, shown in Figure IV-11, is similar to the data value translation module. The differences between the two are as follows:

- The element name translation module operates on atom units containing element names, rather than data values
- The NAD is queried with element name translation functions, rather than data value translation functions
- There is no need to locate corresponding data values, since element name translation is not value relative.

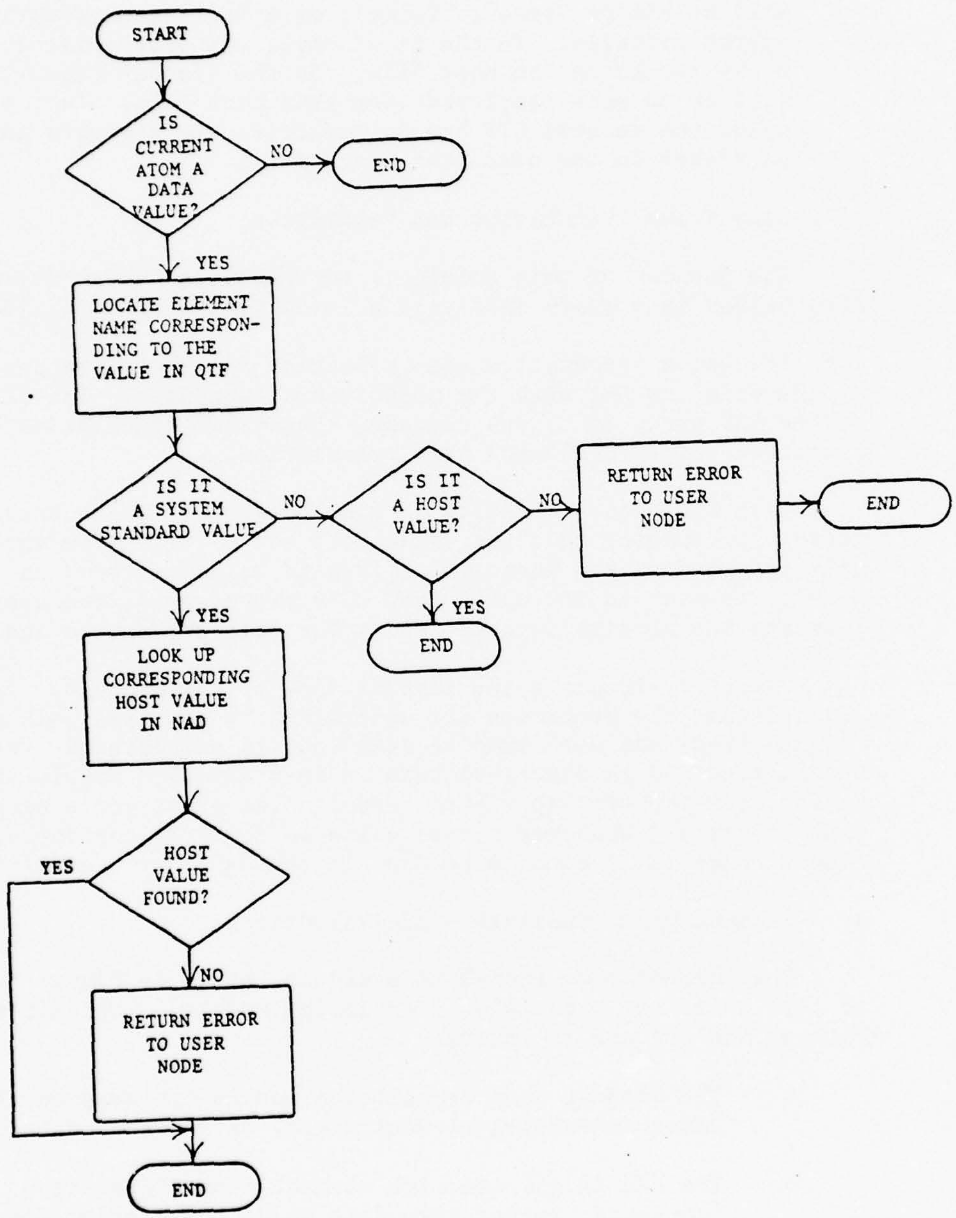


Figure IV-10. QTF Processor, Data Value Translation Module.

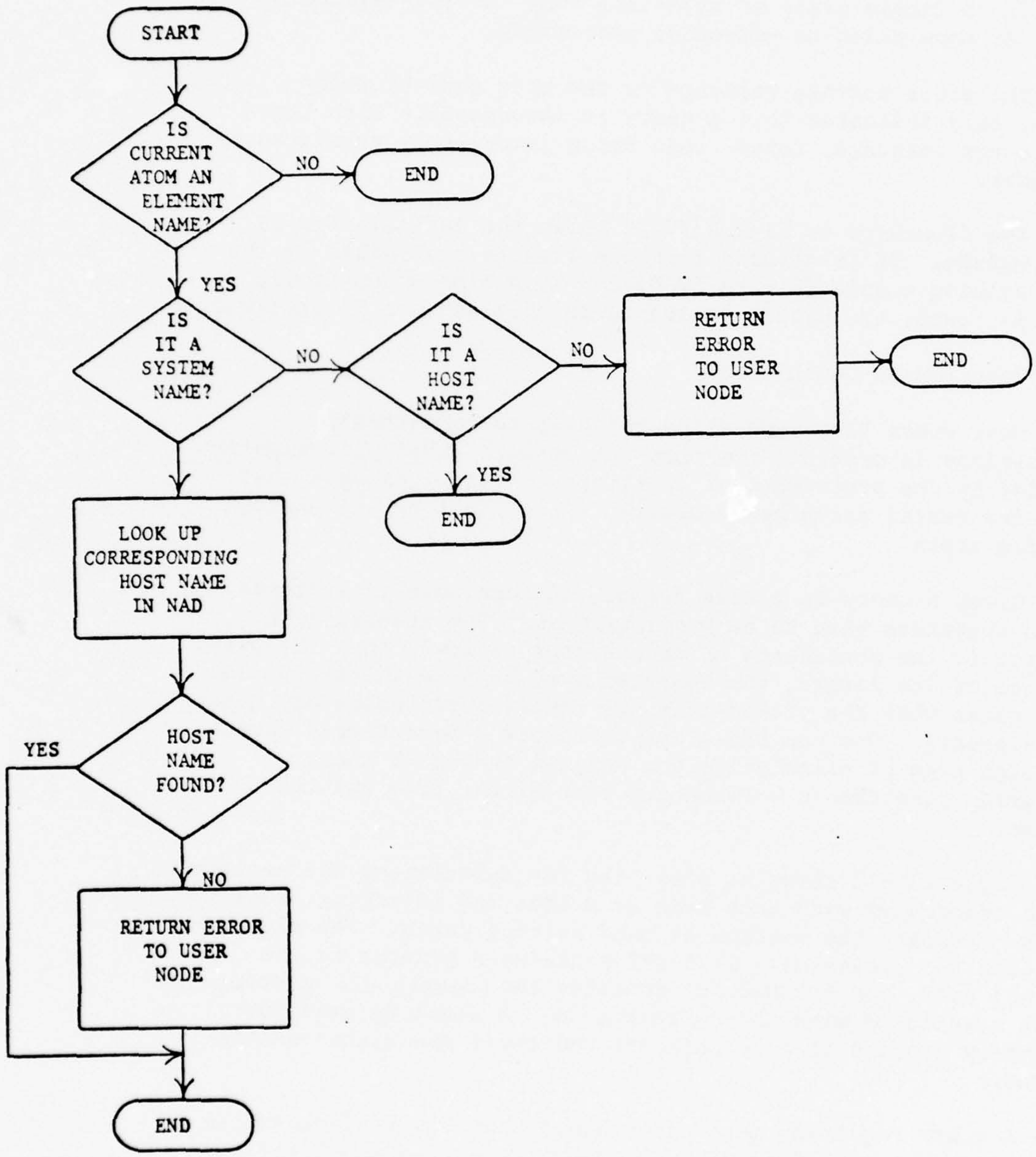


Figure IV-11. QTF Processor, Element Name Translation Module.

e. Operator Existence Test

The Operator Existence Test module checks whether operators in a query exist in the host query language. This module provides a simple means of rejecting some invalid queries at the host node prior to extensive processing.

The error message returned to the user node by this module only indicates that a qeury is incompatible with the host query language, rather than being incorrectly submitted by the user.

The flowchart in Figure IV-12 shows the logical flow of this module. It is assumed that the flow is controlled by the tree walking module (Figure IV-8) and each time a non-atomic node is found, the module described in this section is executed.

f. Parentheses Depth Check

Most query languages allow the user to parenthesize expressions in order to override the default order of evaluation implied by the precedence of operators. Those languages that do allow nested parentheses usually set a limit on the maximum nesting depth.

Given a query in a tree format, one can easily determine which operators need to be parenthesized. The general rule is that if the precedence of an operator is lower than the precedence of its father, the operator must be parenthesized. In some cases when the precedences are equal, parentheses may also be necessary. One can therefore calculate a parentheses count for each tree by calculating the maximum number of nodes requiring parentheses between the root of the tree and its leaves.

Figure IV-13 shows an algorithm for calculating the nesting depth associated with each node of a tree and rejecting those trees exceeding the maximum allowed nesting depth. The algorithm utilizes two variables: CURR-PTR contains a pointer to the current node, and PAREN-COUNT contains the parentheses nesting depth associated with the current node. A stack is used for temporary storage of node pointers and their associated nesting depths.

A query requiring more parentheses than the maximum number allowed can be transformed by applying the Factor and Distribute functions to reduce the number of parentheses. Figure IV-14a shows the use of the Distribute function in parentheses reduction. The left side of the figure requires parentheses to evaluate the OR operator before the AND operator. The expression on the right is equivalent, but does not require parentheses because the evaluation sequence will be correctly deduced from the operator precedence. Figure IV-14b shows how the Factor function can be used in a similar manner to eliminate parentheses.

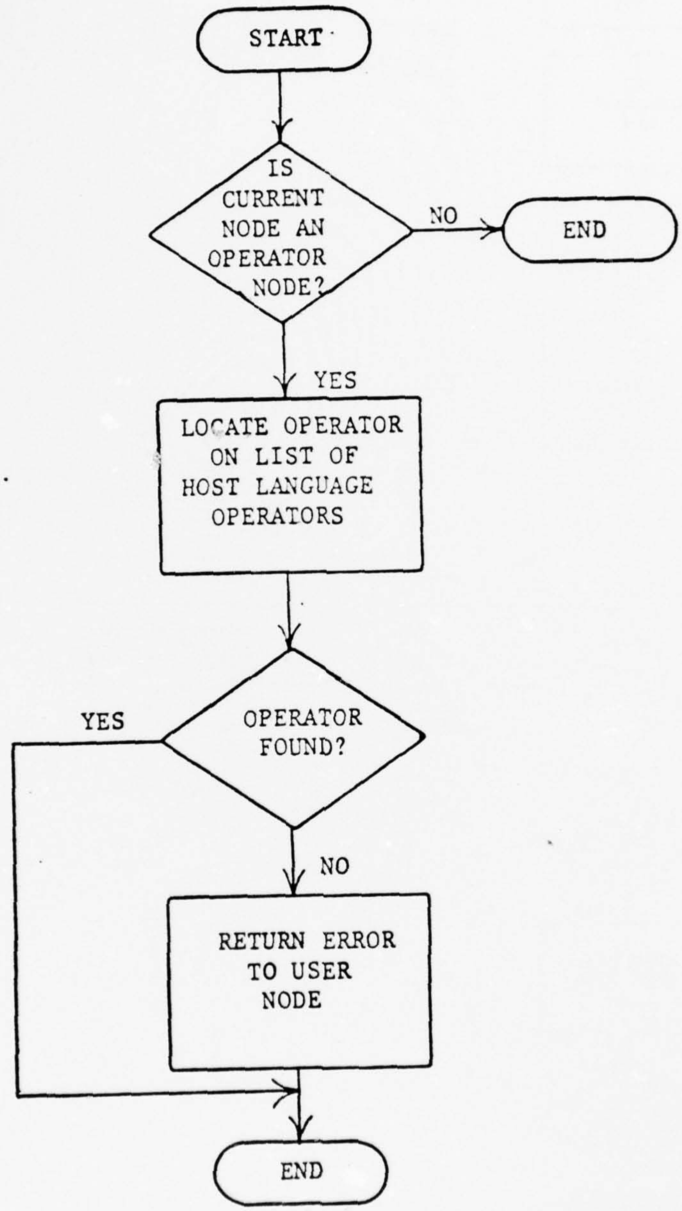


Figure IV-12. QTF Processor, Operator Existence Test Module.

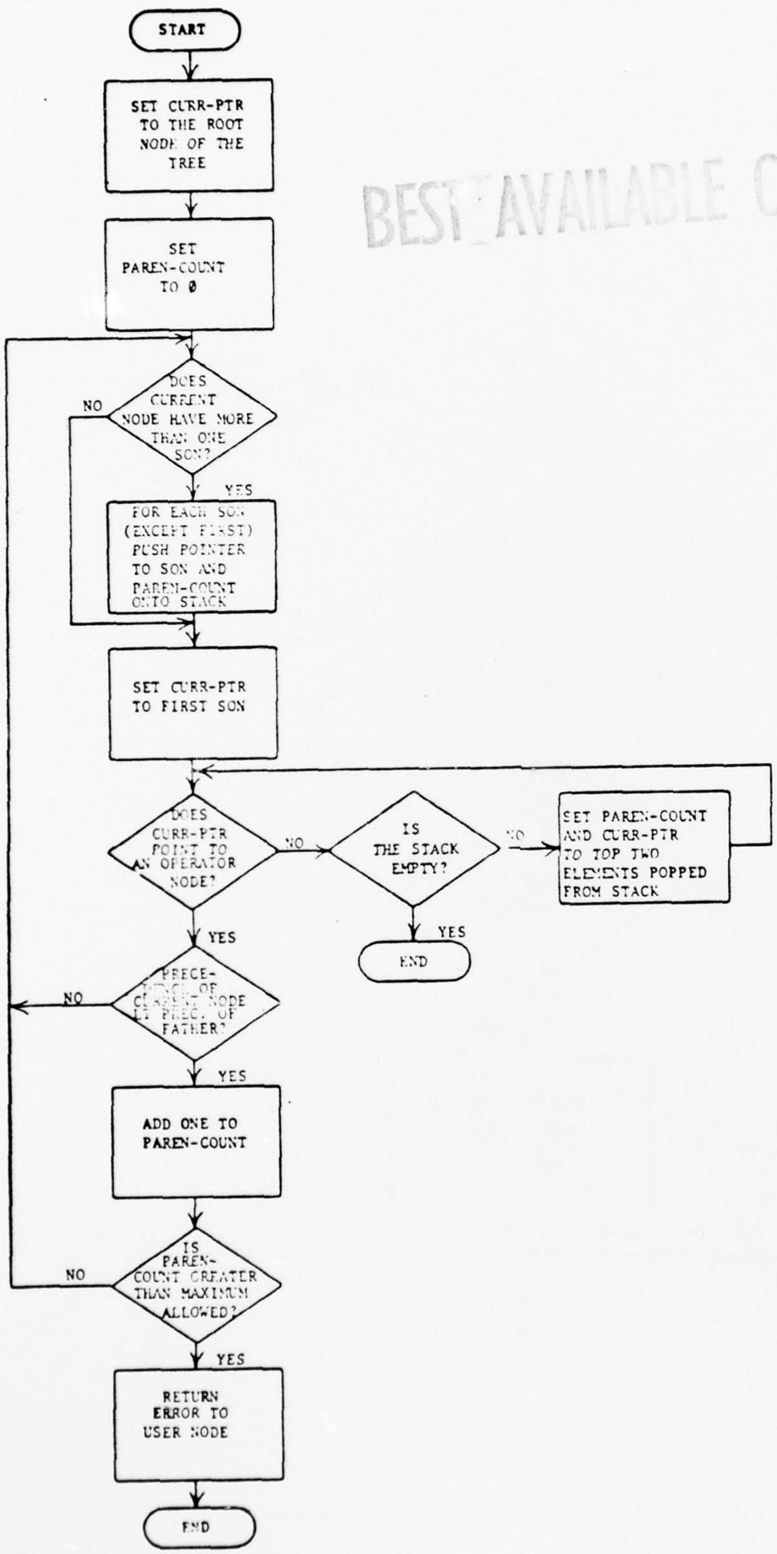
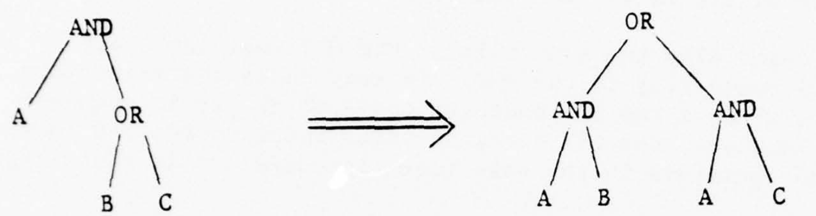
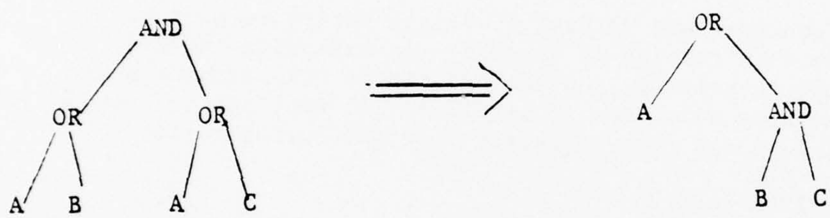


Figure IV-13. Algorithm for Testing for Maximum Parenthesis Nesting Depth.



A AND (B OR C) A AND B OR A AND C

a. Parentheses reduction accomplished by distribution.



(A OR B) AND (A OR C) A OR B AND C

b. Parentheses reduction accomplished by factoring.

Figure IV-14. Examples of Parentheses Reduction.

5. HL GENERATOR

The transformation from the QTF into the host query language, referred to as syntax generation, is performed by the HL Generator. The process is accomplished by a driver program which references a syntax table, described in detail in Section III-5.

The QTF codes play the same role in the QTF that the Query Normal Format (QNF) codes play in the QNF. In many cases the values are the same. However, whereas the QNF contains codes which are independent of the host query language, the QTF contains codes which correspond one-to-one to verbs and operators in the host query language and is thus language dependent.

The Host Language Generator consists of several recursive subroutines. The DRIVER subroutine interprets the syntax table. The CNV subroutine interprets the CNV routine call in the expansion rules. The CNVN routine performs the CNVN call. As additional needs arise, other routines may be defined for use in the syntax table.

The above routines are recursive, so all variables used are reallocated each time each routine in invoked. An exception to this rule is the OUTPUT-BUFFER variable and the associated pointers which are global to all HL Generator routines. This allows all routines to append portions of the host language query directly into the output buffer.

a. The DRIVER Routine

The DRIVER routine initiates the tree expansion, and is called recursively to expand all nodes in the tree. The routine is called with one parameter, a pointer to the node to be expanded. The HL Generation process is therefore initiated by calling the DRIVER with the root node as the parameter.

The DRIVER looks up the node it is processing in the syntax table and interprets the associated expansion rule by moving character strings in the rule directly to the output buffer and calling appropriate routines whenever a routine call is encountered. A flowchart of the DRIVER subroutine is shown in Figure IV-15.

b. The CNV Routine

The CNV routine accepts two parameters: a pointer to the node being processed (CURR-NODE), and a sequential number of the son to be processed (SON-NUM). The parameter SON-NUM=0 indicates that the value contained in the current node is to be retrieved. Otherwise, the routine looks up in the syntax table the precedences of the two nodes involved, inserts a pair of parentheses around the son node if necessary, and invokes the DRIVER program to process the appropriate son node. A flowchart of CNV is shown in Figure IV-16.

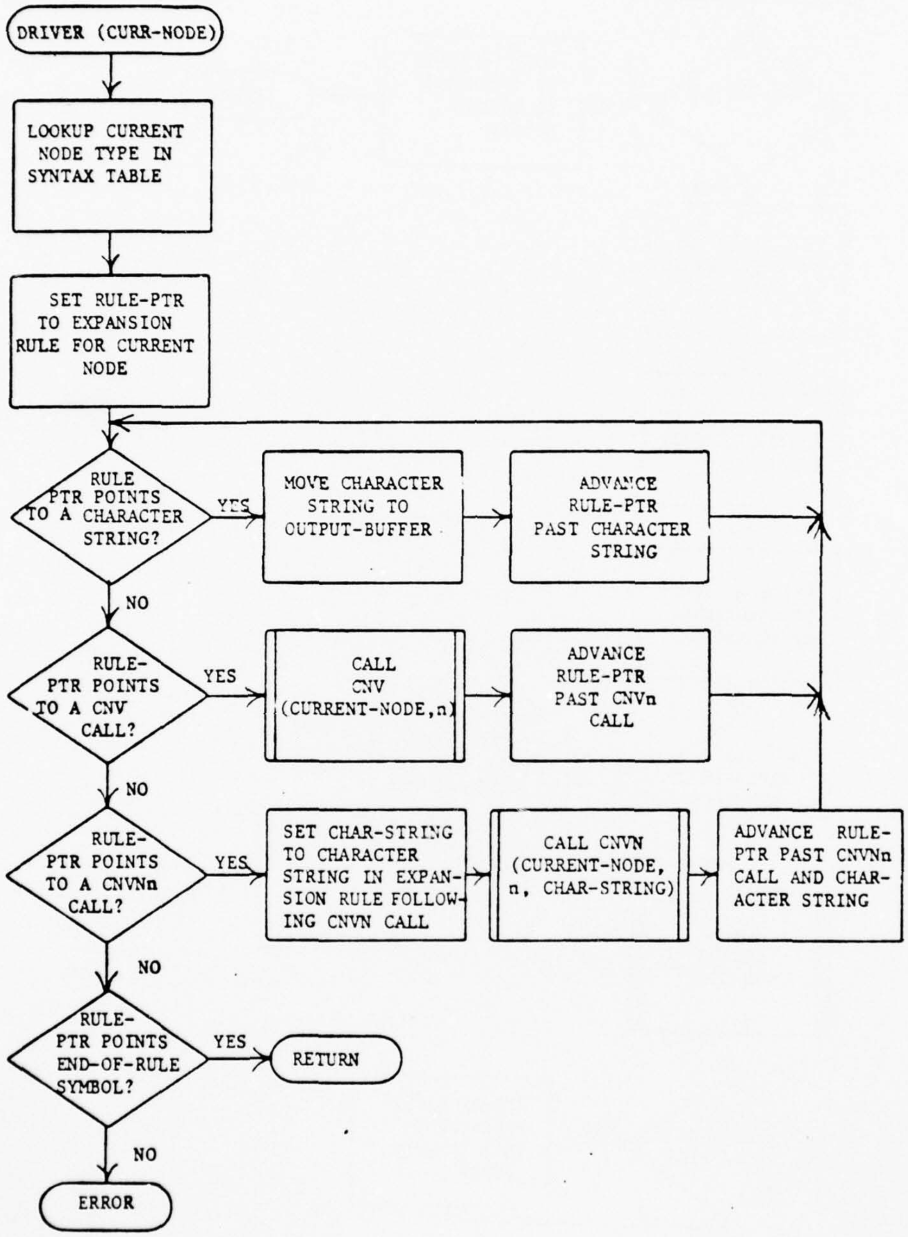


Figure IV-15. The HL Generator, DRIVER Program Flowchart.

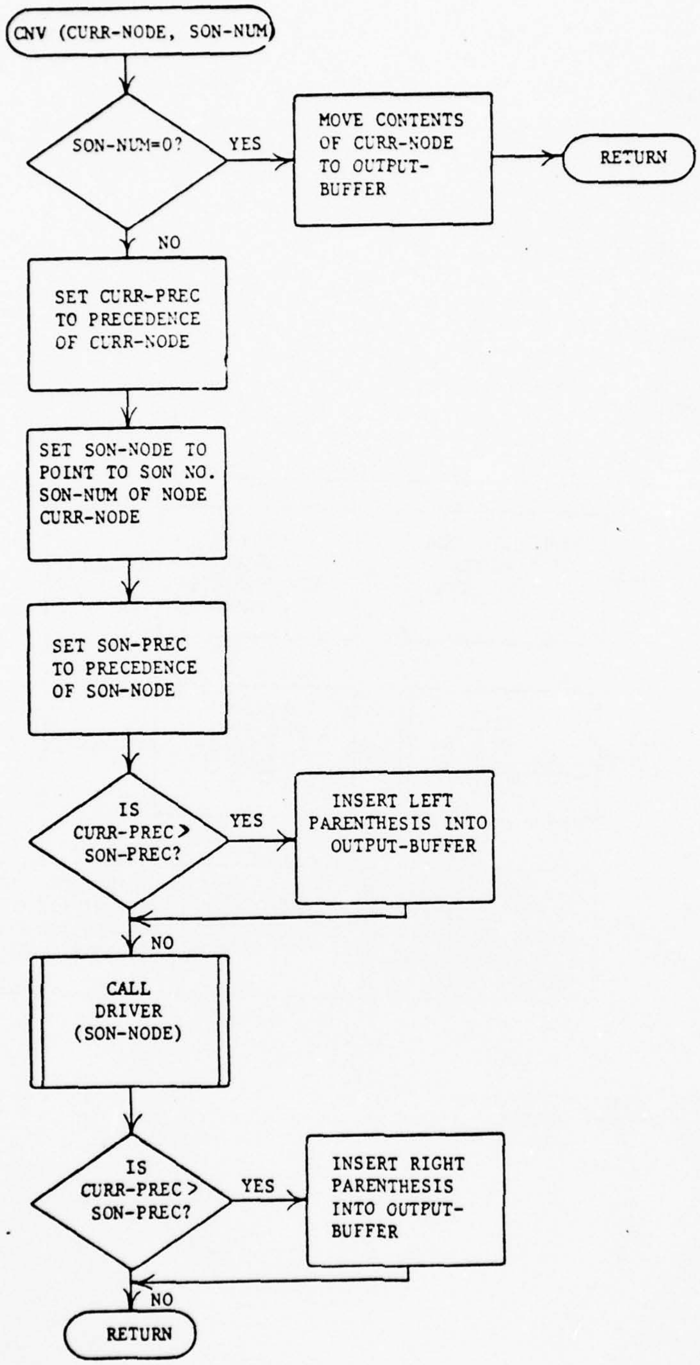


Figure IV-16. The HL Generator, CNV Program Flowchart.

c. The CNVN Routine

The CNVN routine processes a variable number of son nodes. CNVN accepts three parameters. The first are, CURR-NODE, contains a pointer to the current node. The second parameter SON-NUM, contains the number of the first son to be processed. The third parameter points to a character string to be inserted between each pair of nodes processed.

The first son is processed in the same manner as in the CNV routine. If additional sons exist, each one is processed in turn.

The CNVN routine does not allow the SON-NUM parameter to take on the value 0.

A flowchart of CNVN is shown in Figure IV-17.

6. HL PROCESSOR

The HL Processor is a highly query language dependent module and therefore no generalized routines have been designed for it. The HL Processor modules for specific languages are described together with the descriptions of other language dependent modules.

7. *MODULES UNIQUE TO DIAOLS*

This section describes those QTF Processor algorithms that are designed especially for the DIAOLS language. The implementation of these modules will utilize, to the maximum possible extent, primitive tree operations. The resulting modules will be designed to process the DIAOLS language only. The DIAOLS specific algorithms described here together with the general modules described earlier will form a complete Host QIP system for the DIAOLS data base.

a. WITH Operator Processing Module

The WITH logical operator in QNF corresponds to the TO logical operator in DIAOLS which can only be used with other TO operators in a RELATE verb and cannot be combined with AND or OR operators as is allowed in QNF. Thus a special module is necessary to restructure the query to comply with the restrictions on the use of the TO operator in DIAOLS.

A WITH operator in DIAOLS may only operate on relational expressions and other WITH expressions. If that is not true in a QTF, the QTF can be transformed by distributing the WITH operator over AND and OR operators until all resulting WITH operators operate on other WITH operators or on relational expressions.

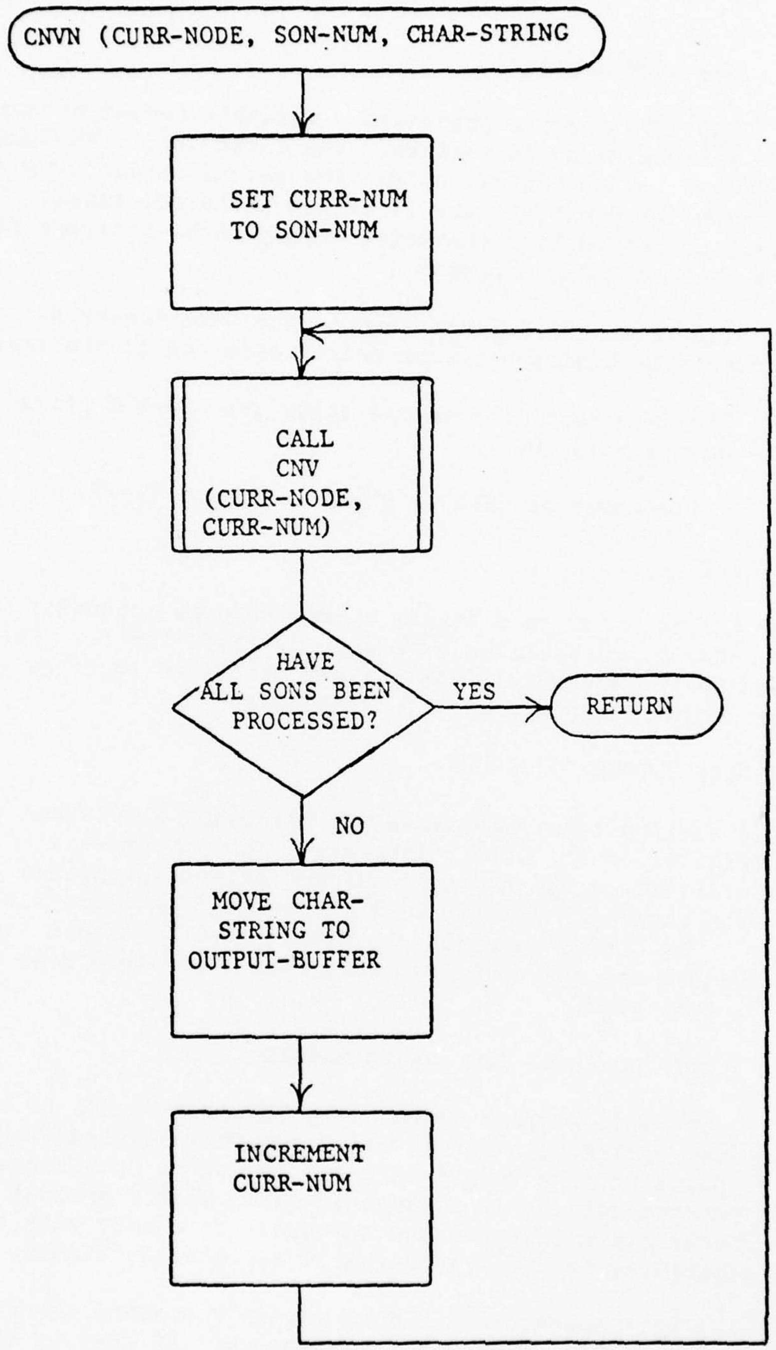


Figure IV-17. Flowchart for CNVN Subroutine in HL Generator Module.

A second restriction states that a WTIH operator in DIAOLS may only operate on relational periodic elements. To comply with this restriciton, all elements in relational expressions operated on by WITH operators are checked. Any WITH operators not operating on relational periodic elements are changed into AND operators.

Validation of the WITH operators in a DIAOLS query is performed in two different passes through the QTF.

The first pass, shown in Figure IV-18a, distributes the WITH operator over other logical operators until all WITH operators operate on relational or other WITH expressions.

The second pass, shown in Figure IV-18b, converts all WITH operators not operating on relational periodic expressions into binary AND operators.

b. Verb Selection

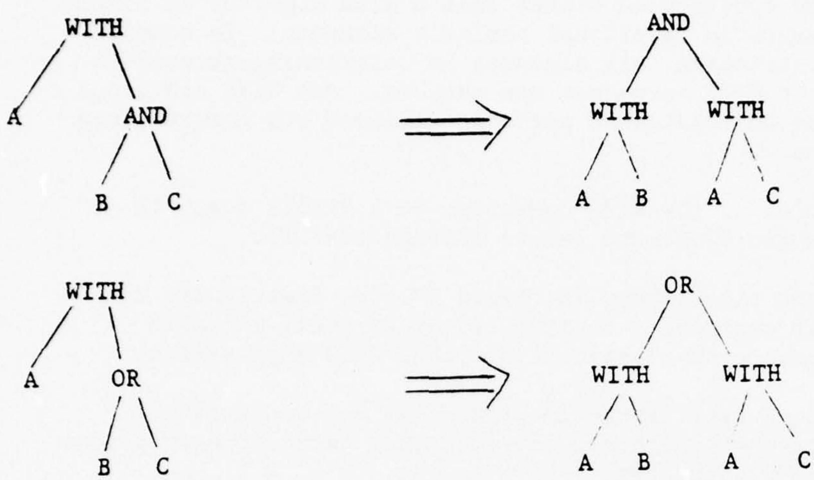
The primitive tree operations described in Section IV-4a can be used in verb selection. This section describes an algorithm utilizing primitive tree operations for verb selection in DIAOLS.

The algorithm operates on a tree structure. It is assumed that all operations are initially structured as a single tree under SELECT retrieval verbs. The goal of the algorithm is to separate such a tree into several branches equivalent to the original tree, each headed by a different verb. Only branches connected with the logical connector AND at the SELECT node can be assigned to separate verbs.

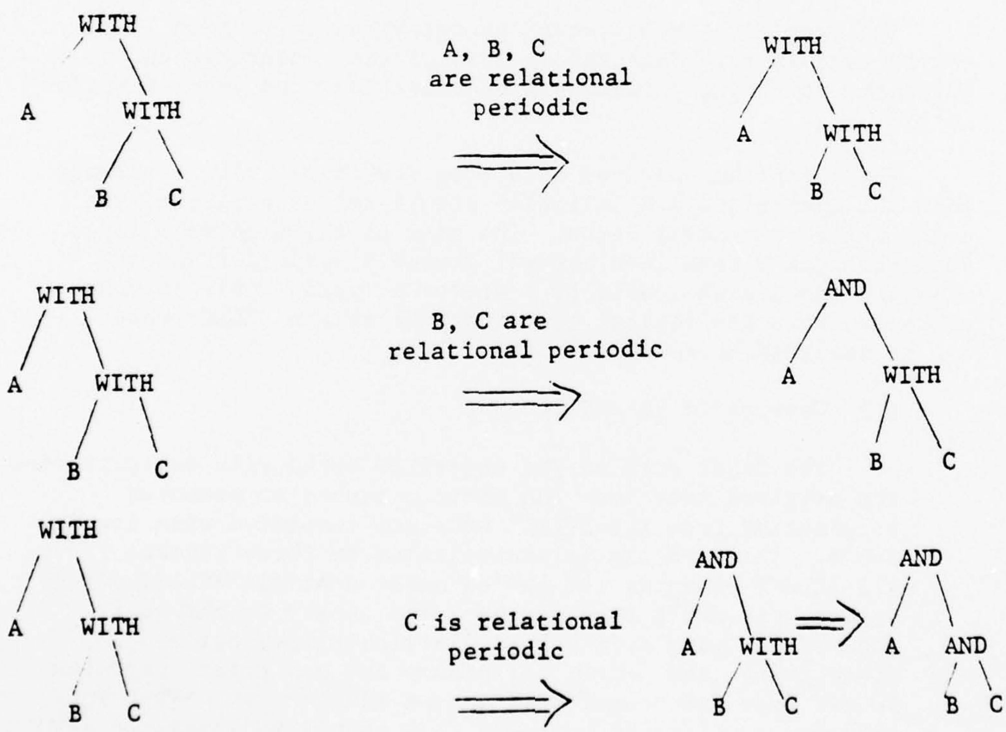
(1) Conversion to AND Groups

The first part of the algorithm deals with restructuring the original tree into the maximum number of branches originating from the SELECT node and connected with logical AND's. This process is accomplished in three stages; first, all SELECT branches are AND'ed under a single SELECT verb; second, and AND's that can be moved closer to the root node by applying a Factor or Distribute operator are transformed; and third, all binary AND connectors near the SELECT node are transformed into a single n-ary AND. The tree now consists of branches that cannot be separated any further for the purpose of verb selection.

Figure IV-19 illustrates how the above algorithm might be applied. Figure IV-19a shows the incoming tree structure. Figure IV-19b shows the same structure after all SELECT verbs have been merged into a single SELECT verb.

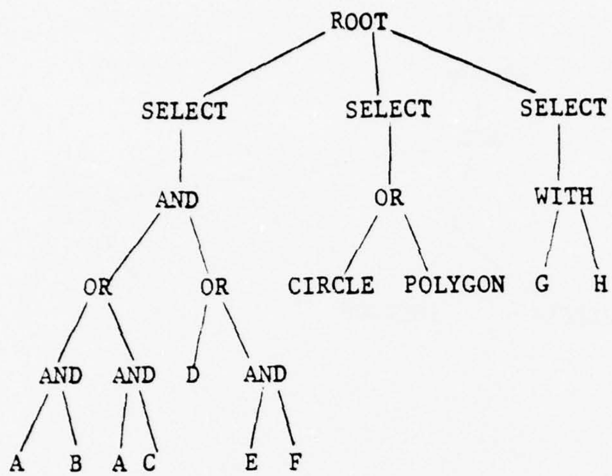


a. Distributing WITH over AND and OR operators.

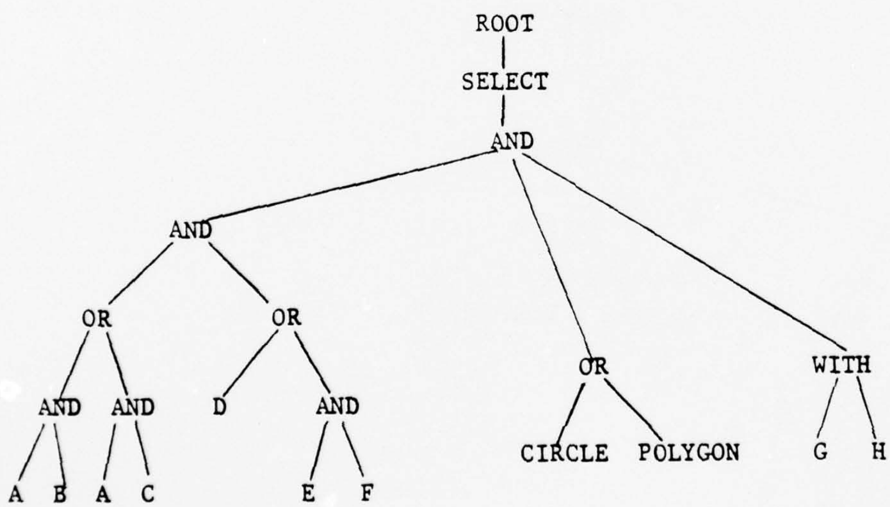


b. Selective conversion from WITH to AND operators.

Figure IV-18. WITH Operator Validation.



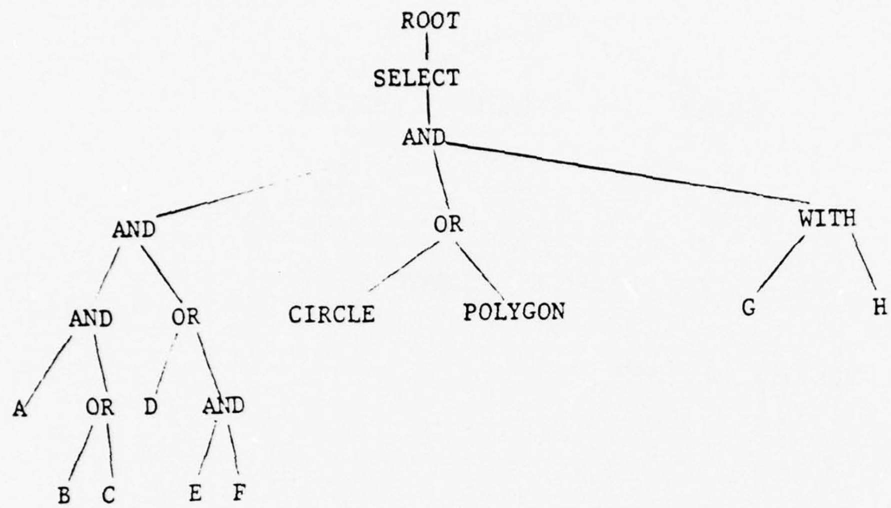
a. Sample query arriving at a DIAOLS data base.



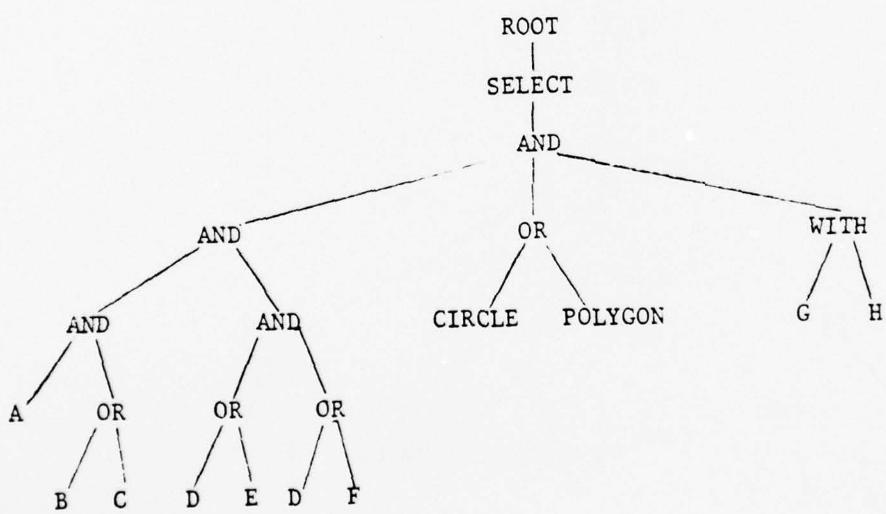
b. Above query after merging all SELECT verbs into a single SELECT verb.

Figure IV-19. Sample Application of the DIAOLS Verb Selection Algorithm. Each atom in this figure (A through H) represents a relational expression.

(Page 1 of 4)



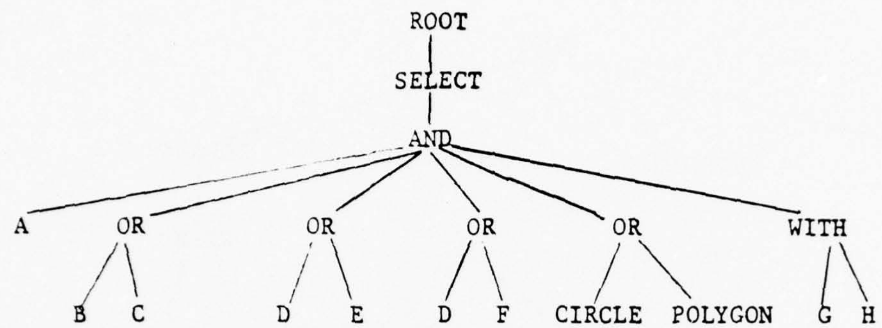
c. Above query after one application of the Factor routine.



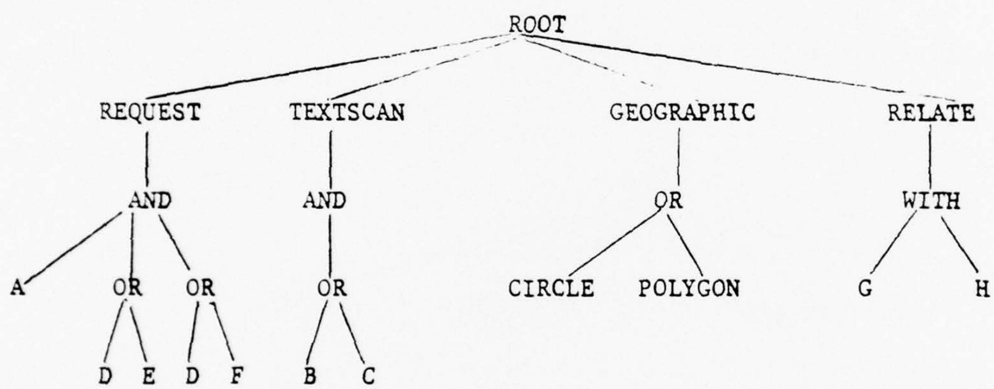
d. Same query, after one application of Distribute.

Figure IV-19. Sample Application of the DIAOLS Verb Selection Algorithm.

(Page 2 of 4)



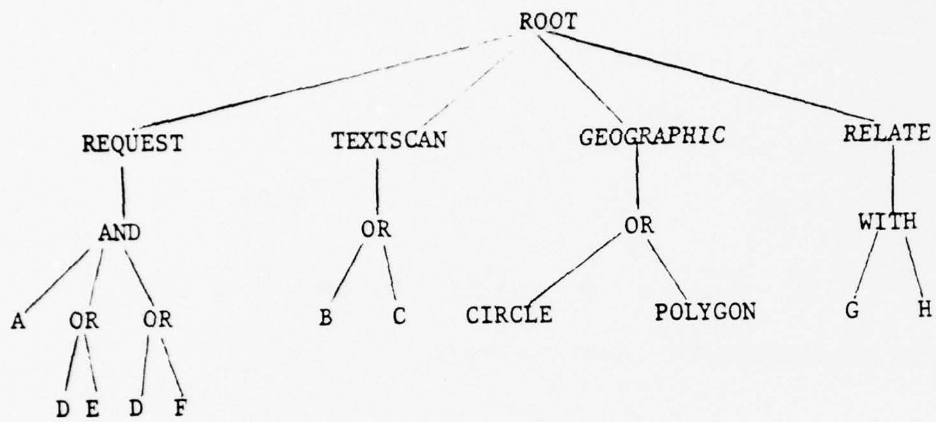
e. Same query after application of Binary to N-ary.



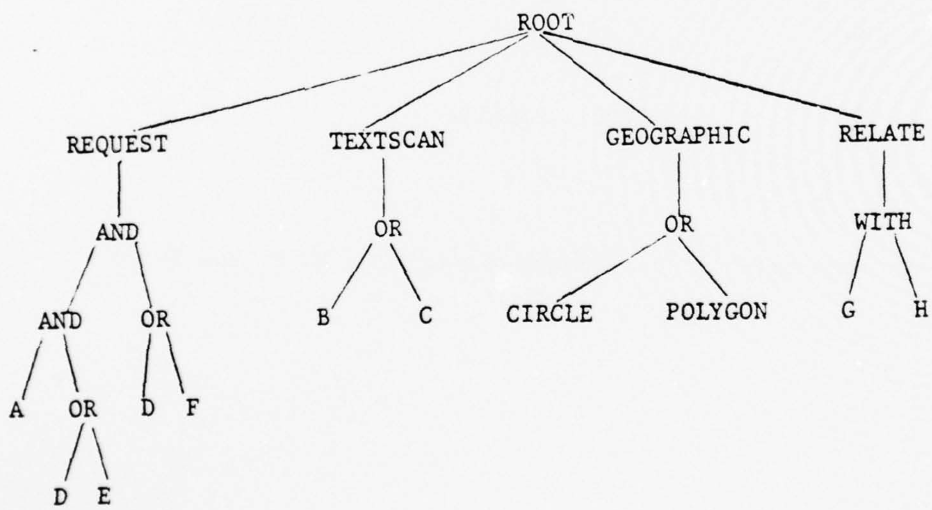
f. Same query after verb selection, assuming elements A, D, E, and F are inverted.

Figure IV-19. Sample Application of the DIAOLS Verb Selection Algorithm.

(Page 3 of 4)



g. Same query after removal of unary AND.



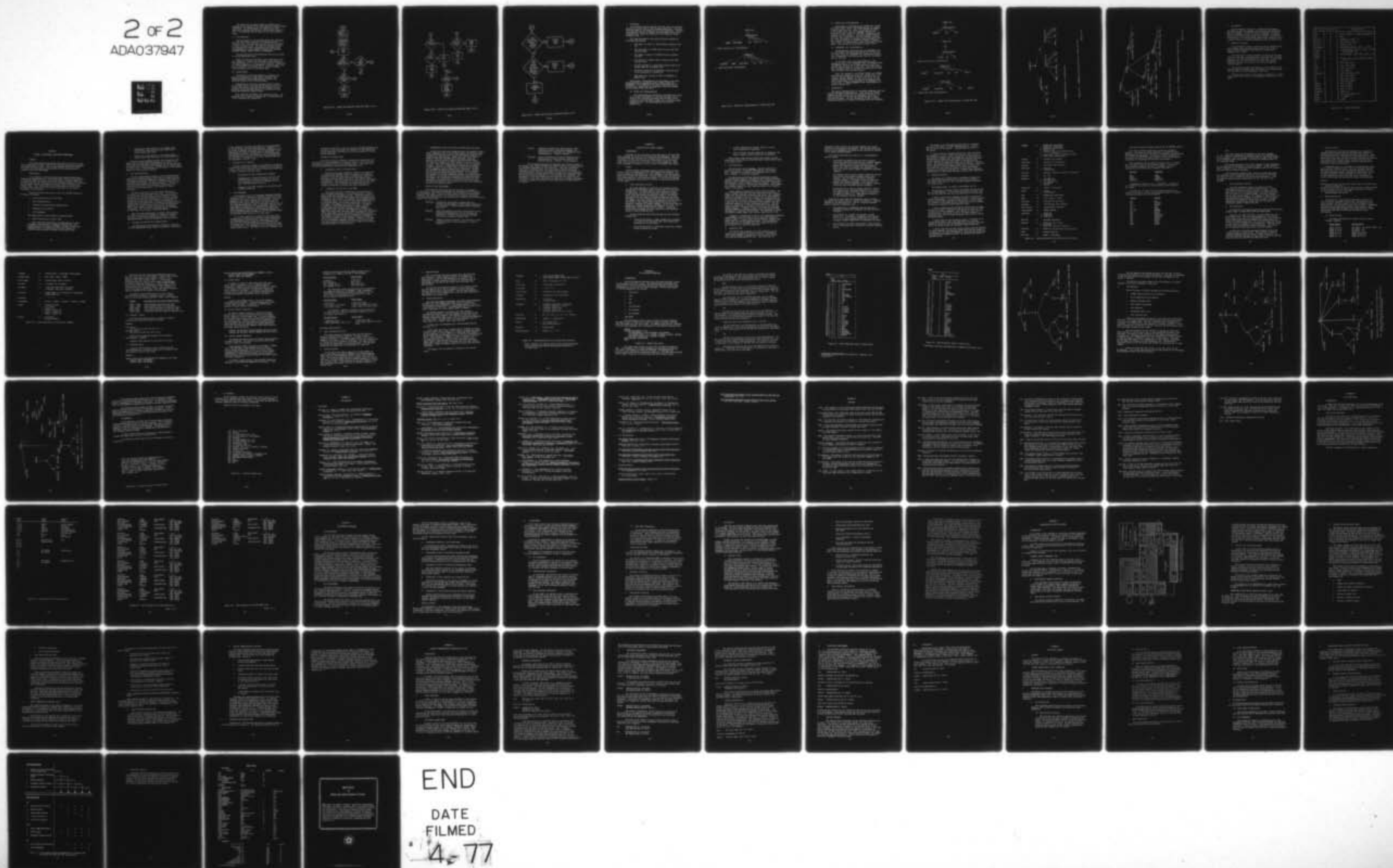
h. Final version of query, generated by applying the N-ary to binary primitive.

Figure IV-19. Sample Application of the DIAOLS Verb Selection Algorithm.

(Page 4 of 4)

AD-A037 947 INCO INC MCLEAN VA F/G 9/2
TRANSPARENT INTEGRATED INTELLIGENCE NETWORK QUERY INTERMEDIATE --ETC(U)
JAN 77 P STYGAR, A PUCHRIK, M TUREK F30602-76-C-0090
UNCLASSIFIED RADC-TR-77-39 NL

2 OF 2
ADA037947



END
DATE
FILMED
4-77

AND connectors are moved towards the SELECT node by applying Factor (Figure IV-19c) and Distribute (Figure IV-19d) primitives. The AND operators are changed to an n-ary operator by applying the Binary to n-ary routine (Figure IV-19e). The resulting tree is ready for the verb selection phase.

(2) Verb Selection

The second part of the algorithm deals with separating the tree into several sections, each associated with a different verb. The algorithm for verb selection in DIAOLS is shown in Figure IV-20. The selection process consists of examining each branch originating at the SELECT verb and assigning it to the appropriate DIAOLS verb (REQUEST, TEXTSCAN, RELATE, CIRCLE, ROUTE, or GEOGRAPHIC).

The above process is straightforward and can be changed to include additional verbs.

Figure IV-19f shows the sample query immediately after verb selection. Figure IV-19f is based on the assumption that expressions A, D, E, and F are all inverted. Additionally, either expression B or expression C, but not both, could also be inverted. A different set of inverted elements would naturally result in a different tree.

(3) Normalization

The purpose of the final phase is to normalize the query by converting all n-ary logical operators into binary operators, since DIAOLS does not permit n-ary operators created by the previous phases.

Normalization of the tree in Figure IV-19f consists of two steps; first, the unary AND is removed since it is invalid and does not serve any function (Figure IV-19g). Secondly, the ternary AND operator is converted into two binary AND operators (Figure IV-19h).

This completes the DIAOLS verb selection process. The resulting tree is ready for another set of transformations of host language query generation.

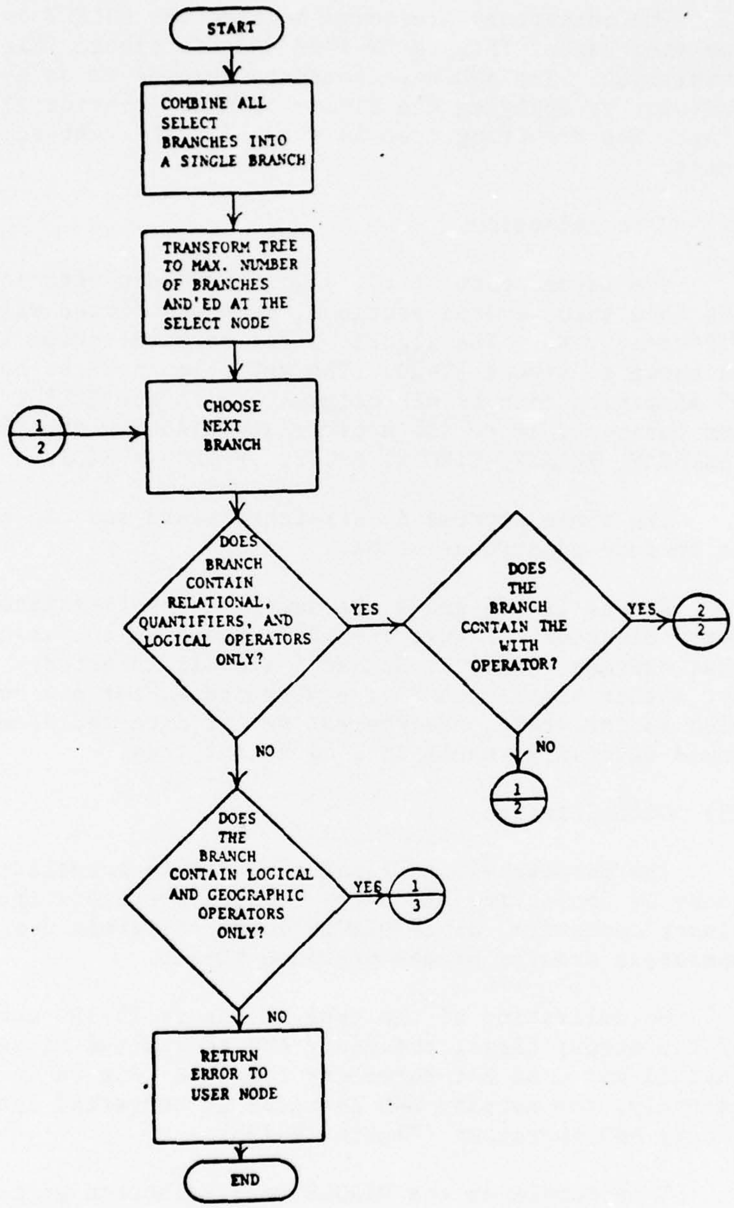


Figure IV-20. DIAOLS Verb Selection Algorithm (Page 1 of 3).

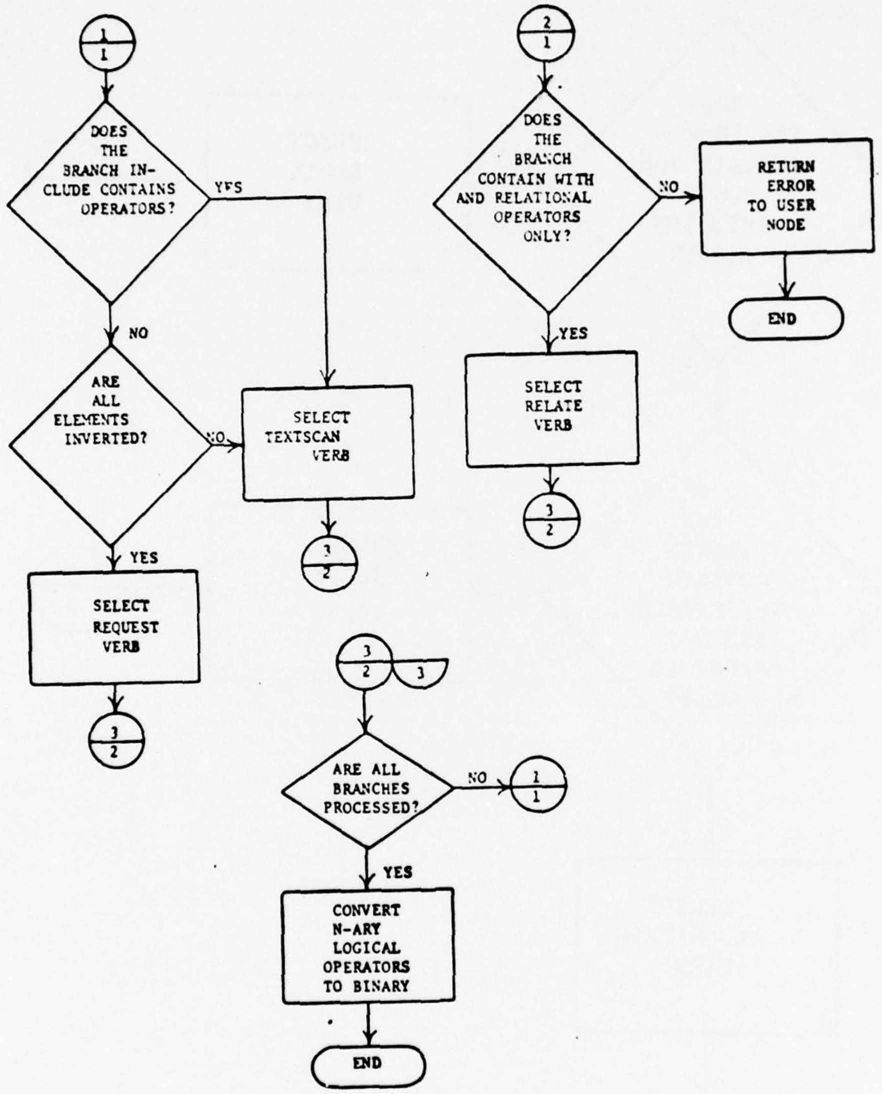


Figure IV-20. DIAOLS Verb Selection Algorithm (Page 2 of 3).

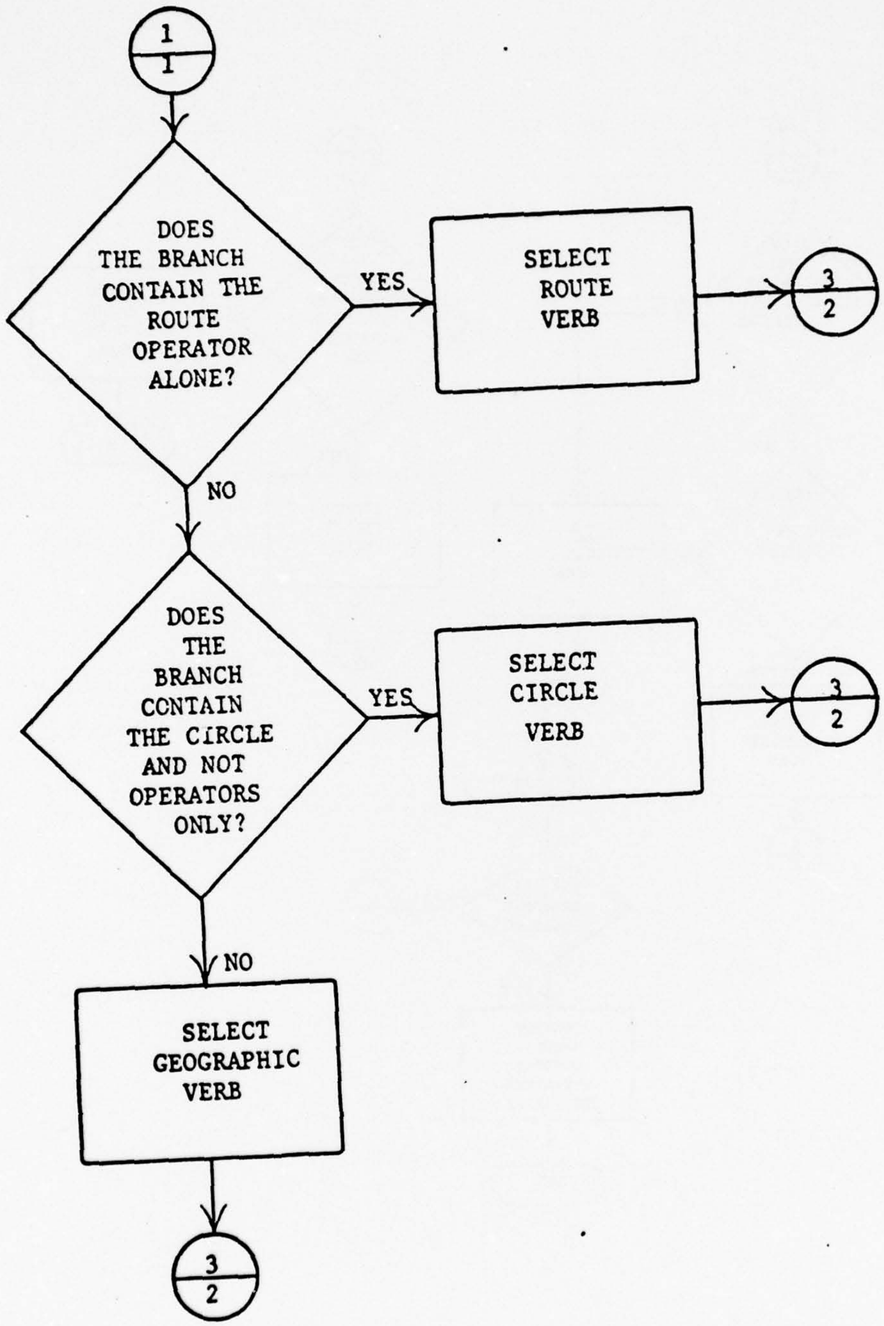


Figure IV-20. DIAOLS Verb Selection Algorithm (Page 3 of 3).

c. Validation

The validation module performs all final tests and transformations of the QTF prior to host language generation. The functions performed by the module fall into two categories: identifying constructs invalid in DIAOLS, and performing certain transformations that cannot be handled by the HL Generator syntax expansion rules.

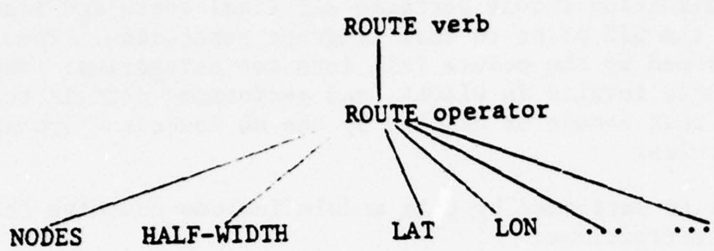
The tests performed by this module include checking the following restrictions:

- The number of nodes in a ROUTE cannot be greater than 506
- The half-width of a ROUTE cannot be more than 2000 nautical miles
- The number of nodes in a POLYGON cannot be greater than 506
- The radius of a CIRCLE cannot be greater than 2000 nautical miles
- The two operands of a relational operator must be an element name and an element value
- Relational operators and geographic operators must not be connected by a logical OR
- Each query must include at least one TEXTSCAN or REQUEST verb.

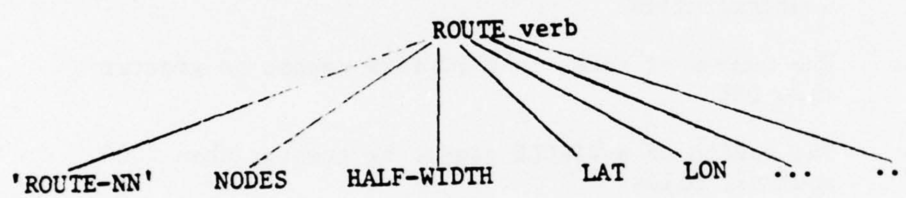
Additionally, transformations are performed on the three verbs pertaining to geographic areas, CIRCLE, ROUTE, and GEOGRAPHIC. It is important that these transformations are coordinated to assure that all geographic areas are assigned distinct names within each query.

(1) ROUTE verb transformations

The transformations performed on the ROUTE verb consist of two steps, the removal of the ROUTE operator and the addition of an operand containing the area name, which consists of the word ROUTE, a dash, and a two-digit number assigned by QIP. The transformations are shown in Figure IV-21.



a. ROUTE verb prior to transformation.



b. ROUTE verb after transformation.

Figure IV-21. ROUTE Verb Transformations in DIAOLS Host QIP.

(2) CIRCLE verb transformations

Transformations performed by the CIRCLE verb consist of several steps. First, the CIRCLE operator is removed from the tree structure. Next, two operands are added to the CIRCLE verb. The first operand contains the character string 'INSIDE' or 'OUTSIDE', depending on the presence of a NOT operator. The second operand contains the area name, consisting of the word 'CIRCLE' followed by a dash, followed by a two-digit number assigned by QIP. Next, the NOT operator is removed if present. Finally, the operands are resequenced to comply with DIAOLS requirements. This process is illustrated in Figure IV-22.

(3) GEOGRAPHIC verb transformations

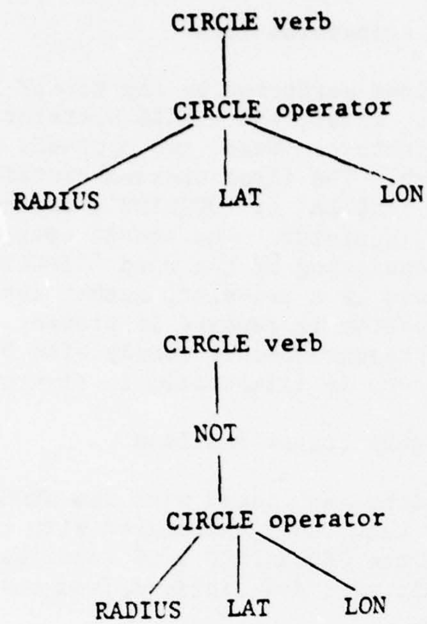
Transformations associated with the GEOGRAPHIC verb are more complex than those associated with the previous two verbs, since the GEOGRAPHIC verb requires the separation of geographic area descriptions from the conditional part of the tree.

As the first step of the transformation, a new operand is added to the GEOGRAPHIC verb, and AREALIST node. The AREALIST operator has an AREA operand for each geographic area description in the query. Each AREA operator has a variable number of operands, consisting of the area name followed by the area description.

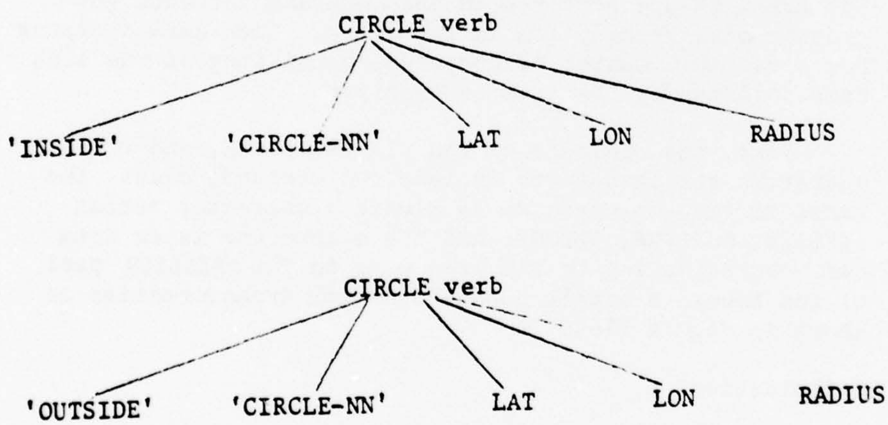
Next, the operands of the CIRCLE, ROUTE, and POLYGON operators are changed to include two operands each. The first of the two operands is always a character string (INSIDE, OUTSIDE, ALONG), and the second one is an area name corresponding to the area name in the AREALIST part of the tree. A sample GEOGRAPHIC verb transformation is shown in Figure IV-23.

d. Optimization

The initial implementation of the DIAOLS translator will not include optimizing transformations. Even though the DIAOLS language includes several sets of equivalent constructs, the user's guide does not provide any information about their relative efficiency. As such information becomes available, optimization modules can easily be inserted into the translator.

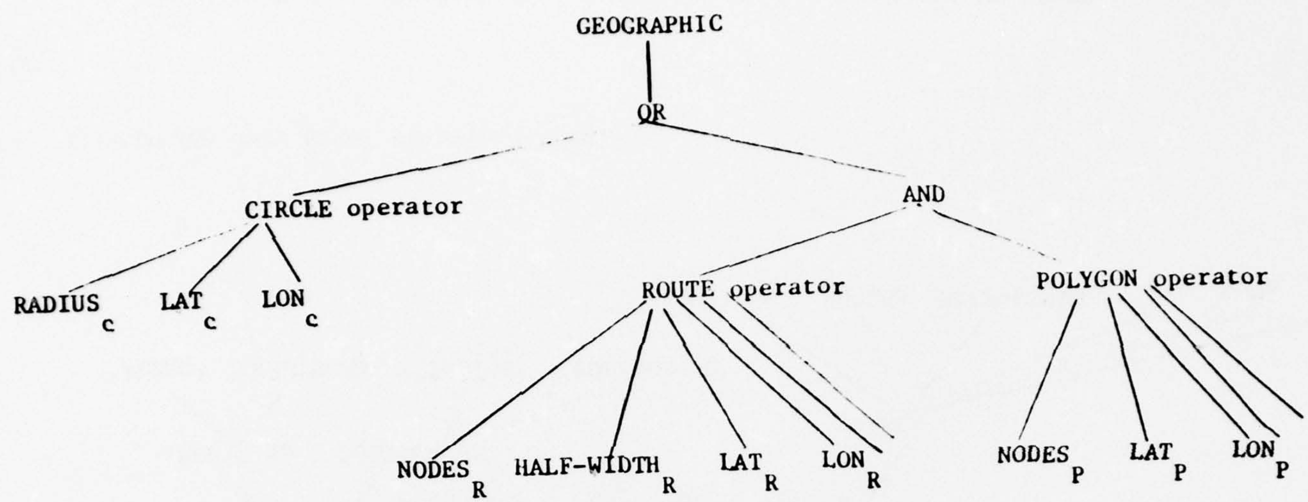


a. CIRCLE verb prior to transformation.



b. CIRCLE verb after transformation.

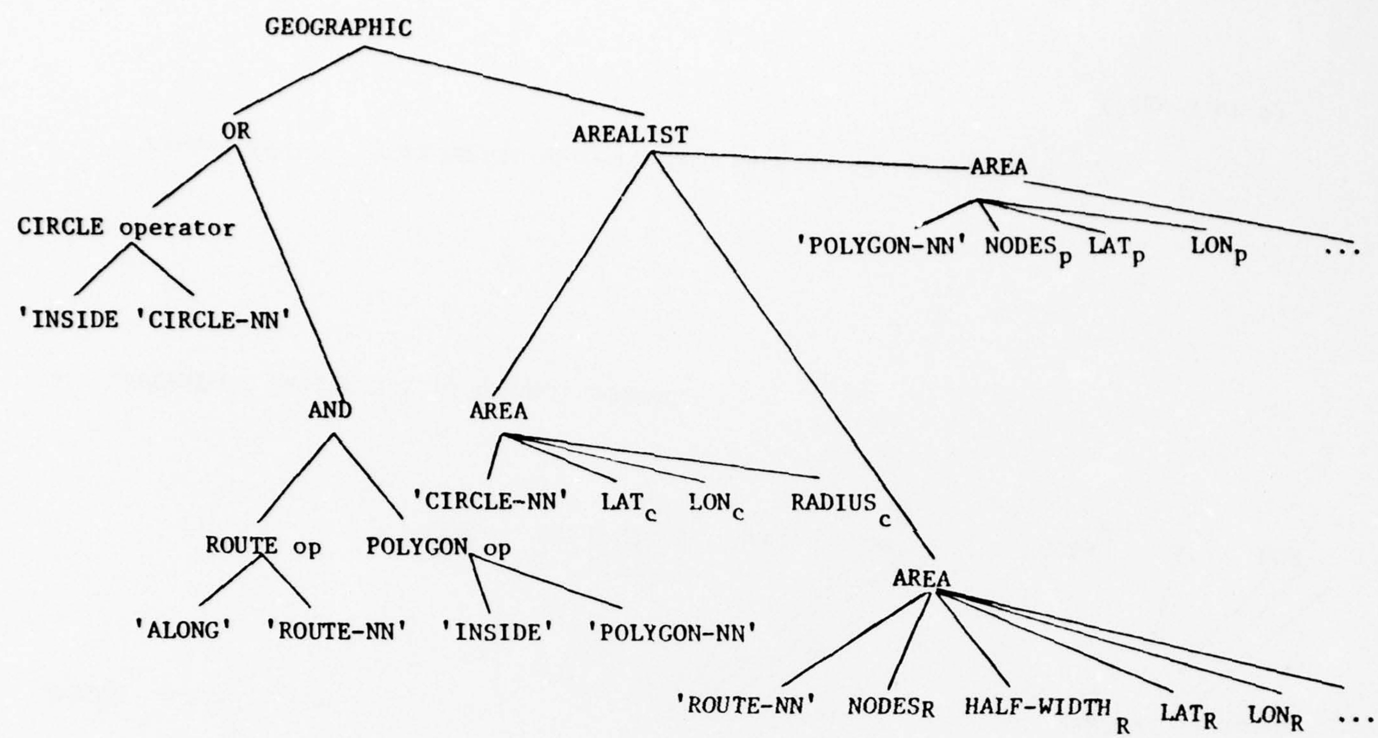
Figure IV-22. CIRCLE Verb Transformation in DIAOLS Host QIP.



a. GEOGRAPHIC verb prior to transformation.

Figure IV-23. GEOGRAPHIC Verb Transformations in DIAOLS Host QIP.

(Page 1 of 2)



c. GEOGRAPHIC verb after transformation.

Figure IV-23. GEOGRAPHIC Verb Transformation in DIAOLS Host QIP.

(Page 2 of 2)

e. HL Generator

The HL Generator consists of a general Driver program, applicable to any host language, and a snytax table describing the host language. The Dirver program interprets the syntax table and generates a query in the host language. A detailed description of the Driver program can be found in Section IV-5. A description of the Syntax Table format exists in Section III-4. A syntax table for the DIAOLS language is shown in Figure IV-24.

f. DIAOLS HL Processor

The HL Processor accepts a character string containing the host language query, performs three scans of the string to insert certain characters, and passes the finished host language query to QDNC.

The first pass of the HL Processor locates words that must begin on a new line, and substitutes a line feed and a carriage return for the space preceding each of the words. Words always beginning on a new line include: all verbs, the logical connectors AND, OR and TO, the word IF, and anything that follows a period.

The second pass checks the lengths of lines created by the previous pass and changes additional spaces to carriage returns and line feeds to split excessively long lines.

The final pass inserts a line number, consisting of a three-digit number and a space, at the beginning of each line of the query.

QTF Code	Precedence	Expansion Rule
ROOT	Ø	'FILE,␣',CNVN1,'␣','␣END.␣DISPLAY␣BYE.'#
FILE	Ø	CNVØ#
REQUEST	Ø	'REQUEST␣IF␣',CNV1,'.'#
TEXTSCAN	Ø	'TEXTSCAN␣IF␣',CNV1,'.'#
RELATE	Ø	'RELATE␣',CNV1,'.'#
GEOGRAPHIC	Ø	'GEOGRAPHIC,␣NEW␣IF␣',CNV1,'.␣',CNV2#
CIRCLE (verb)	Ø	'CIRCLE,␣NEW␣IF␣',CNV1,'␣',CNV2,'␣',CNV2, '␣EQ␣',CNVN3,',␣'#
ROUTE (verb)	Ø	'ROUTE,␣NEW␣IF␣',CNV1,'␣EQ␣',CNVN2,',␣'#
PRINT	Ø	'COMPOSE,␣NEW,␣1␣DATA␣=␣SELECT␣',CNVN1, '␣','.'#
OR	1	CNV1,'␣OR␣',CNV2#
AND	2	CNV1,'␣AND␣',CNV2#
WITH	2	CNV1,'␣TO␣',CNV2#
NOT	3	'NOT␣',CNV1#
CONTAINS	4	CNV1,'␣CONTAINS␣',CNV2#
GT	4	CNV1,'␣GT␣',CNV2#
LT	4	CNV1,'␣LT␣',CNV2#
EQ	4	CNV1,'␣EQ␣',CNV2#
CIRCLE	4	CNV1,'␣',CNV2#
ROUTE	4	CNV1,'␣',CNV2#
POLYGON	4	CNV1,'␣',CNV2#
AREALIST	4	CNVN1,'␣'#
AREA	4	CNV1,'␣EQ␣',CNVN2,',␣'#
VALUE	5	':',CNVØ,':'#
NAME	5	CNVØ#

Figure IV-24. DIAOLS Syntax Table.

SECTION V

SUMMARY, IMPLICATIONS, AND FUTURE DEVELOPMENT

1. SUMMARY

This Final Report has presented the results of the work on the Query Intermediate Processor (QIP) of the proposed Transparent Integrated Intelligence Network (TIIN). This section presents some of the implications of the QIP design and the relation of the current effort to possible future TIIN development.

2. IMPLICATIONS

It is difficult to speculate about what usage patterns might evolve in an implemented TIIN. The design effort has been concerned with the differing needs of the host nodes which seek to minimize update and storage costs, the user nodes which seek to optimize retrieval and analysis, and the network itself which seeks to optimize overall efficiency and response time. Thus, the design effort has attempted to provide flexibilities for the user nodes, for the host nodes, and for the future development of the network.

There are several broad areas in which the TIIN/QIP design has an impact, as follows:

- Ease of transition to use of TIIN
- File standardization
- Authority for network data administration
- Expansion of the network
- User languages.

The implications in these areas are discussed below.

a. Ease of Transition to Use of TIIN

TIIN offers the possibility of standardization of query languages, standarization of communications protocol, and standardization of network data bases. However, because file standardization is a complex and difficult problem, the TIIN/QIP design includes two modes of data base access:

- Transparent access applies to the element names and/or value codes which have been standardized across several data bases

- Direct host access applies to the element names and/or value codes which have not been standardized.

The direct access mechanism offers the data analyst the possibility of using one query language (e.g., TAP) for all hosts, and using one common communications discipline, but with the option of continuing to use a familiar set of element names and value codes. This minimizes the TIIN impact on the existing library of file description manuals.

b. File Standardization

The file standardization effort in the intelligence community is an on-going activity which aims to achieve uniform conventions for data representation throughout the network of intelligence data bases. The use of uniform conventions will facilitate data analysis and cross-correlation. The use of uniform conventions for all processors would enhance the efficiency and responsiveness of the network in crisis situations.

One basic objective of the TIIN system has been to facilitate the file standardization effort. This objective is reflected in the QIP design in the transparent access mechanism which provides substantial user control of naming conventions in the rest of the network. This provides the possibility of developing and testing new conventions for element names and value codes, by using the transparent access mechanism for a limited set of users, while the operational needs of the network continue to be met via the existing conventions. The use of new conventions may be introduced to user nodes in an orderly, controlled fashion.

When it becomes appropriate to change a file structure at a host node, it may be possible to reduce or eliminate the impact on TIIN users through the techniques of data mapping available in the QIP transparent access mechanism.

c. Authority for Network Data Administration

The TIIN design raises questions relating to authority for data administration and network coordination. Maintenance

of the integrity of network data bases is the responsibility of the individual data base administrators. However, it is not clear, at present, who should have the responsibility for arbitrating the needs of various users of the network. Data administration involves the mediation of conflicting needs and applications at both the node-level and the system-level. The host DBA is responsible for security considerations and the assignment of privacy locks.

d. Expansion of the Network

New host nodes may be added to the network in an orderly step-by-step process which eases the impact on the network as a whole. Each step provides a discrete operational capability to form the basis for the next step:

- Installation of TOSS communication software
- Implementation of those portions of the host QIP concerned with the special features of the host query language, the host error messages and the host report generation interface
- Mapping of host data elements into standard system element conventions.

e. User Languages

The TIIN/QIP project is developing language processing techniques which are primarily intended to convert one user language to any one of a set of host languages. The actual design of a user query language will be the goal of the TIIN/TAP effort, and in that effort the basic issues of user language preferences, ease of training, ease of use, and user/analyst productivity will be addressed. However, user nodes have the flexibility to develop an alternate user query language by substituting an alternate processor for the TAP subsystem which accesses all the QIP features simply by producing a DRIF version of the user query.

It is then possible to evaluate the alternate query language with a limited group of users, while concurrently and independently the real world requirements continue to be met on the existing system with no requirements for retraining and with no departures from normal system usage and maintenance. Two languages could be compared with "all other factors equal." The choice of a query language could

be based on factors at a user site rather than DBMS implementation factors at a host site where the complexity of the operational system places many constraints on the actual design of the user access procedures.

3. ADVANCED TIIN CAPABILITIES

This report has been primarily concerned with specification of capabilities to be implemented in a prototype system. This report also has several references to "later development stages." These and other possible post-prototype capabilities are summarized below.

a. Transparent Access to Multiple Hosts

The multi-host query capability refers to the capability to automatically transmit queries to several host files as a result of one user query, and to automatically combine the responses to form a single response to the original query. This capability has many facets and is described in more detail in Appendix J. This capability is the main focus of the TIIN research effort, and is perhaps the single most important capability necessary to achieve data base transparency.

b. Integration of TIIN and a Local DBMS

This report has been concerned only with the issue of query language transparency in distributed access to heterogeneous data base management systems. Transparent access is required for all the other DBMS capabilities, including report preparation, data update, file merging, and programmed access to retrieved data. It is possible that these capabilities can be provided through the existing capabilities of the local host DBMS or the DBMS at the local user node. Integration of TIIN and a local DBMS would have the effect that the analyst would use the query language, report language, programmed access, and the other capabilities of the local DBMS, and would access other host DBMS as transparent extensions of the local DBMS. In the interests of efficiency it would be desirable to implement such an integration by modifying the local DBMS to achieve a graft between TIIN and the DBMS so that the two systems can grow together. Though no host DBMS in the intelligence community is available for such a graft, the concept may have possibilities with a DBMS on the network interface computer at the local node.

c. Transparent Access to IDS-Style and DBTG-Style Data Base

In this report the basic assumption about the network standard query language is that it is oriented towards hierarchical files because most large data bases in the intelligence community and elsewhere are hierarchical. Though there are an increasing number of IDS-style and DBTG-style data bases, these are generally smaller and more complex than hierarchical files, and there is a tendency for these systems to become saturated in a short time since usage grows quickly to fill the existing capacity. The use of a hierarchical oriented query language in TIIN is based on the belief that the majority of the data in the community will be hierarchically structured. However the IDS-style and DBTG-style data bases must not be excluded from the post-prototype development stages. It is possible and desirable to provide transparent access to non-hierarchical data bases. Such access can be provided by partitioning such data bases into a collection of hierarchical files with cross references. The substructure of multiple hierarchical files are the prime targets of user queries, with the cross references handled on an exception basis.

4. STATUS OF TIIN DEVELOPMENT

The subsystems comprising TIIN are described in Appendix G of this report. The overall development and implementation of TIIN will be accomplished under a series of specification projects and development projects. The design projects are concerned with research, functional and preliminary specification, and specification validation for the following areas of the TIIN system:

TIIN/TILF (Transparent Intelligence Language Facility) Creation and maintenance of data element directories for TIIN nodes. (Final Technical Report completed June 1976.)

TIIN/QIP (Query Intermediate Processor) Specification of host independent representation of user queries, and conversion of queries into host query languages. (Final Report completed November 1976.)

TIIN/TAP (Terminal Access Procedures) Specification of user-oriented methods of query expression. (Status: proposed.)

TIIN/RN (Response Normalization) Specification of host independent report formatting languages, and specification of techniques for transparent value standardization. (Status: proposed.)

TIIN/QDNC (Query Distribution and Network Communications) Specification of query routing, tracking, and exercise modules. (Status: to-be-proposed.)

The development projects are organized into distinct stages of development, each stage representing a definite advance in system capacity and attendant enhancement in user capability. The first development stage will be achieved under the to-be-proposed TIIN/QIP Prototype Development project which will implement the results of the design projects to achieve a software system which converts user queries expressed via TAP into queries acceptable to a current DBMS such as DIAOLS. The development of the TIIN system has been organized into highly modular components, so that small development efforts will be needed to implement the special language processing for each additional host DBMS added to the network.

APPENDIX A

DESCRIPTIONS OF QUERY LANGUAGES

1. INTRODUCTION

This part of the Final Report describes some of the host DBMS query languages for which the TIIN system has been designed. Where the information has been available the language descriptions have included a description of the file structures, a description of the retrieval verbs, and a description of the protocol for query submission. These query languages are presently in use to query intelligence data bases.

2. DIAOLS

DIAOLS (Defense Intelligence Agency On-Line System) provides access to over 100 intelligence files via Honeywell 635 hardware. The file structures are relatively flat, with a maximum of two hierarchical levels and no inter-file references. Several search verbs are available, specialized relative to some structural aspect of the file, specialized relative to geographic-area selection criteria, or specialized relative to text scans.

a. Query Submission Protocol

The query language is available in an interactive mode through ISS (Intelligence Support System) and in a batch mode through COINS. In either mode the user formulates a query by typing each verb and each selection criteria on separate lines, with line numbers automatically typed by the system. The line numbers are used for editing purposes and for error references. In the interactive mode the user may submit only one command (verb plus criteria) per execution, and must verify the result of each command before submitting the next verb phrase in a sequence of related commands. In the batch mode the sequence of related commands may be submitted as a unit. In either mode a query may be saved for re-use. A query may contain query variables which are replaced by user specified values at query submission time.

In the interactive mode (ISS) the user has the following capabilities:

- o Using the NEW option, a query command may be entered in a line-by-line mode with each line automatically prefixed with a line number.
- o Using the OLD option, a previously saved query command may be accessed for re-use.

- o A query command may be edited, saved for re-use, and/or submitted for execution.

- o After the query has been submitted for execution, the status, summary, and/or results may be obtained.

When typing a query across several input lines, the only restriction is that the only term which may be split across lines is the search value.

b. Data Structures

The basic data unit is element. The data types for an element are numeric, formatted, alphanumeric, and standard date. Elements may be named in two synonymous ways; short name (4 digits), or long name (mnemonic).

The data structure options include hierarchical records (called periodic elements), inversion files (called retrievable elements), groups (physically contiguous files scanned as a unit), and subfields. DIAOLS supports random and indexed - sequential file storage structures. Records are fixed-length: each element has a maximum length, and each repeating element has a maximum number of occurrences.

DIAOLS supports hierarchical records with two levels. The level 1 elements have one value per record while each level 2 element may have multiple occurrences per record, with a maximum number of occurrences which may be different for each element. The N-th occurrence of a level 2 element is associated with the N-th occurrence of another level 2 element by literally storing a subscript (N) with each occurrence of each element, and such subscript elements are called "relational periodic." The subscript is transparent to retrieval verbs and is used only for update purposes. A periodic set is a collection of relational periodic elements. A periodic subset is an occurrence of data conforming to a periodic set structure. The subscript is called the periodic subset number (PSS) and may range from 01 to 99.

A group is a combination of from 2 to 13 physically contiguous elements into a data structure which is named and referenced like an element. A group cannot contain periodic elements, and DIAOLS does not provide repeating groups. Group elements may be queried only in a TEXTSCAN verb, only using the operator HAS to search for a specified substring.

c. Retrieval Verbs

Each search verb produces a hit file consisting of the records which satisfied the search criteria. Every search verb (except REQUEST) considers only the records in the hit file from the preceding search verb. Thus the effect of

consecutive search verbs is to conjunct together the criteria specified in the separate verb phrases. The user may reduce the amount of file scanning by including as many criteria as possible in the initial REQUEST phrases.

Four search verbs deal with basic (i.e., non-geographic) search criteria.

- The REQUEST and REFINE verbs are used to specify selection criteria which mention only inverted elements (i.e., elements which have an inversion file). The REFINE verb uses the hit file from the preceding search verb, while the REQUEST verb applies to the whole file and is used to begin a sequence of retrieval commands.

- The TEXTSCAN verb is used to specify selection criteria which involve non-inverted elements and/or group elements. The selection criteria may reference inverted elements. Relational periodic elements are treated like periodic elements. All values of the element are tested independent of any implied association with other relational periodic elements.

- The RELATE verb is used to specify selection criteria which mention only relational periodic (i.e., level 2) elements, where the criteria must be satisfied for the same occurrence (i.e., same implicit subscript) of the relational periodic elements.

Three search verbs deal with geographic search criteria. One operand is usually implicit, namely, the position - defining elements in the specified file. The user specified operands serve to describe a geographic area.

- The operands for a GEOGRAPHIC verb may specify an arbitrary combination of polygons, circles, and route corridors.

- The operands for a CIRCLE verb specify a single circle (i.e., a latitude, a longitude, and a radius). The result of the distance calculation is stored in a (pseudo) element named 9999 for subsequent retrieval.

- The operands for a ROUTE verb specify a route corridor (i.e., a half-width and a sequence of latitude-longitude pairs).

The radius of a circle and the half-width of a corridor must be in the range .1 to 2000 (nautical miles). A route may have from 2 to 506 points. A polygon may have from 3 to 506 points.

A route corridor or polygon side may not cross the Greenwich Meridian. In such a case, the route must have a node on the Greenwich Meridian; and the polygon must be subdivided into polygons with a common side lying along the Greenwich Meridian. A route corridor must not cross either of the two meridian poles. There is no theoretical limit to the number of circles, routes, and polygons which can be entered at one time. Therefore, when a polygon or route exceeds the 506 limit, the areas can be divided into more than one polygon or route and chained together by the logical connector OR. Response time increases significantly as conditional expressions become more complicated.

d. Syntax Description

The syntax for a DIAOLS query is presented in Figure A-1. This syntax description ignores the restrictions imposed by the line-by-line protocol. The following comments supplement the formal syntax.

The maximum number of levels of parentheses is six.

The optional relational operator qualifiers ALL and NOT may not be used together. If they are, then both are ignored, that is, "ALL NOT" and "NOT ALL" are both equivalent to a no-op word.

A search value must be enclosed in colons(:) and may not exceed 56 characters. For a TEXTSCAN the limit is 200 characters. Two successive colons represent a blank value. A search value must conform to the format and the coding scheme for the element to which it is applied at query execution time.

A query variable must be enclosed in double quote (") marks and may not exceed 15 characters, and may not be used more than 32 times in a query statement. This feature is useful for prestored queries. The actual value for the query variable is specified by the user at query submission time through a system initiated dialog.

A dollar sign ($) may be used in lieu of a character in an actual search value or the value specified for a variable operand at execution time to designate which character positions are to be ignored during the designated compare operation.

A search value may be split across as many lines as necessary. The first character following the blank after the line number on a continuation line will be appended to the character string from the previous line immediately after the character preceding the carriage return.

```
<command>          ::=  REQUEST [IF] <select-expr>.
                      | REFINE [IF] <select-expr>.
                      | TEXTSCAN [IF] <select-expr>.
                      | RELATE <to-expr>.
                      | GEOGRAPHIC [<pos>] [IF]<area-expr><map>
                              <def-list>
                      | CIRCLE [<pos>] <area-relation><node>, radius.
                      | ROUTE [<pos>] <map><route-def>

<to-expr>          ::=  <criteria> [TO <to-expr>]

<select-expr>      ::=  <and-expr> [OR <select-expr>]

<and-expr>         ::=  <paren-expr> [AND <and-expr>]

<paren-expr>       ::=  <criteria>
                      | (<select-expr>)

<criteria>         ::=  <element> [<qualifier>] <relation> <value-list>

<qualifier>        ::=  ALL | NOT

<relation>         ::=  [IS] EQUAL [TO]
                      | [IS] LESS [THAN]
                      | [IS] GREATER [THAN]
                      | [IS] BETWEEN
                      | CONTAINS

<value-list>       ::=  <value> [, <value-list>]

<value>            ::=  :string:
                      | "query-variable"

<pos>              ::=  element-name, element-name

<map>              ::=  1|2|3, number |4, <node>

<area-expr>        ::=  <and-area> [OR <area-expr>]

<and-area>         ::=  <area-criteria> [AND <and-area>]

<area-criteria>    ::=  <area-relation> <area-name>

<area-relation>    ::=  INSIDE | OUTSIDE

<area-name>        ::=  CIRCLE-<id>
                      | ROUTE-<id>
                      | POLYGON-<id>

<def-list>         ::=  <area-def> [<def-list>]

<area-def>         ::=  CIRCLE-<id> <node>, radius.
                      | <route-def>
                      | POLYGON-<id> node-count <node-list>.

<route-def>        ::=  ROUTE-<id> node-count half-width <node-list>.

<node>             ::=  latitude, longitude

<node-list>        ::=  <node> [, <node-list>]
```

Figure A-1. Syntax Description for DIAOLS Retrieval Verbs.

Two search values are always required for the BETWEEN operator.

More than one search value may be used with the EQUAL operator. The string of values specified in conjunction with EQUAL means that any one of the values is acceptable for satisfying the retrieval requirements. The string of values specified in conjunction with NOT EQUAL means that any value not specified is acceptable for satisfying the retrieval requirements.

Each coordinate used to define a circle, route, or polygon may be entered in one of the following formats. If any part of a coordinate (degrees, minutes, or seconds) contains a leading zero, that zero must be entered by the user.

Latitude	Longitude
ddx	dddx
ddmmx	dddmmx
ddmmssx	dddmmssx
(x=N or S)	(x=E or W)

A geographic identifier (i.e., a POLYGON-id, a CIRCLE-id, or a ROUTE-id) may use one or two characters (usually digits) as a label.

The words IF, IS, THAN, TO may be included and have no effect on the query. The use of a synonym in place of a keyword will have the same effect as the keyword.

Synonym	Keyword
AND	WITH
HAS	CONTAINS
GT, >	GREATER
LT, <	LESS
EQ, =	EQUAL
BT, BTW	BETWEEN
REQ	REQUEST
REF	REFINE
GEO	GEOGRAPHIC
TEX	TEXTSCAN
REL	RELATE
CIR	CIRCLE
ROU	ROUTE
ALONG	INSIDE

3. TILE

TILE (TIPS Interrogation Language) is the query language for TIPS (Technical Information Processing System) at the NSA (National Security Agency). TILE is available in batch mode through COINS as well as in interactive mode through TIPS, via Univac 494 hardware under the RYE operating system.

One interesting aspect of the TILE language is that parentheses are not permitted for grouping of selection criteria. Several special logical operators (e.g., ALSO) have lower precedence than OR and in effect allow one level of implicit grouping.

The TILE search capability has a variety of substring tests. More important, TILE provides the capability to use the hit file from one search to search a different file for related information. These two features require special consideration for an "all host" query language.

a. Query Submission Protocol

The query submission protocol is based on the job submission procedure of the RYE Operating System. A three line header is entered before the query, to specify the user name, the host site, the program name of the query processor, the job priority, the terminal number, the user's org/code, and the user's phone number. All commands in TILE consist of a verb identifying the statement followed by a string of words describing the command and ending with a period. A single command may occupy several lines, and fully describes all information needed to execute the verb.

b. Data Structures

A collection of contiguous equal-size fields may be associate together under a common name called the family name.

There are two types of files: a single-format file and a multi-format file. The field configuration for all records in the file is the same in a single-format file, while different field configurations (called formats) are allowed within the records in a multi-format file. For both file types all records are the same size (maximum is 495 characters). The first field of each record specifies the format identifier. The record key is whichever field name occurs in all the formats.

A collection of records associated together to describe a single entity is called group. The selection criteria in a search statement serve to access a single record when the retrieval verbs EXT and EXTI are used, while a group is accessed when the retrieval verbs EXTG and EXTIG are used.

c. Retrieval Verbs

The function of the extract statement is to request that information be extracted from the data base and to specify selection criteria for selecting records for extraction. There are two extract verbs: EXT (extract) retrieves only the first qualifying record; EXTI (extract inclusive) retrieves all records that qualify; EXTG (extract group) retrieves only the first qualifying group of records; EXTIG (inclusive extract group) retrieves all groups that qualify.

A dependent search is a sequence of related extract statements querying different files. The first statement extracts some data from a file in the data base and creates a hit file. The second statement extracts information from another file in the data base based on data contained in the hit file. The second statement is tied to the first one by a reference to the hit file created the last time the first file was queried.

Example:

Extract all records from file PER1 having JONES in the NAME field.
    EXTI/PER1; NAME(JONES).
Extract from file PER2 all records that have SS# contents matching the SSN contents of any of the records developed as a result of the last extract on the file PER1.
    EXTI/PER2/PER1; SS# IS SSN.

d. Syntax Description

The syntax for a TILE query is presented in Figure A-2. Only the extract statement is described. The analysis of the print statement will be included in the TIIN Response Normalization study. The description given below is brief to supplement the formal syntax description without being redundant. Many features are described informally here rather than in a formal syntax due to the lack of sufficient information in the user manuals.

(1) Value Criteria

The range specification for search values is like a "between" operator.

Range Example	Alternate Form
GRADE (n TO m)	NOT GRADE < (n) AND NOT GRADE > (m)
GRADE (* TO m)	NOT GRADE > (m)
GRADE (n TO *)	NOT GRADE < (n)
GRADE (n +- m)	GRADE (n-m TO n+m)
GRADE (n + m)	GRADE (n TO n+m)
GRADE (n - m)	GRADE (n-m TO n)

```
<command>          ::=   <extract-verb> / file-name; <select-expr>.

<extract-verb>     ::=   EXT | EXTI | EXTG | EXTIG

<select-expr>      ::=   [<select-expr> ALSO] <or-expr>

<or-expr>          ::=   [<or-expr> OR] <and-expr>

<and-expr>         ::=   [<and-expr> AND] [NOT] <criteria>
                       |  <and-expr> [AND] NOT <criteria>

<criteria>         ::=   element name [IS] [<relation>] (<value-expr>)
                       | element-name [()]

<relation>         ::=   < | > | : | .

<value-expr>       ::=   string | <range> | <format> | *string | string*

<range>            ::=   string TO string
                       | * TO string
                       | string TO *
                       | number +- number [%]
                       | number + number [%]
                       | number - number [%]

<format>           ::=   ?[<format>]
                       | string [<format>]
```

Figure A-2. Syntax Description of TILE Extract Command.

Note that there are unfortunate ambiguities when the key word TO is part of a value search. TITLE (ALPHA TO BETA) denotes a range search (from "ALPHA" to "BETA"), not a search for the value "ALPHA TO BETA". To solve this problem brackets ([]) may be used in place of parentheses to delimit a character string, e.g., TITLE [ALPHA TO BETA]. This artifice must be used whenever reserved characters or reserved words occur in search values. This prevents full use of TILE search capabilities, since many search functions cannot be denoted in the context of the brackets.

The partial value specifications for search values provide several varieties of substring tests. A question mark (?) in a value string indicates a "don't care" character.

Example	Indicated test on value of GRADE element.
GRADE . (ABC)	Left-adjusted value ends with "ABC".
GRADE : (ABC)	Value contains "ABC" as a substring.
GRADE (ABC*)	First three characters of field are "ABC".
GRADE (*ABC)	Last three characters of field are "ABC".
GRADE (?ABC)	A character value has "ABC" as characters 2-4.

(2) Geographic Search

The range test allows the user to specify a range of latitudes and longitudes forming a rectangle.

Examples:

Point Search
    LAT (39 40 12 N) AND LON (078 45 22 W)
Rectangle Search
    LAT (39 TO 40) AND LON (078 TO 079)

There is also a special operator W/IN to permit a circle search. Example:

LAT/LONG (W/IN 121NM OF 25 36 10N AND 075 45 20W)

(3) Existence Search

A special feature allows a user to extract records containing data in a specified field, and to exclude records that contain no data in that field.

Examples:

Extract records where SERVICE=NAVY and REMARKS is not blank
    SERVICE (NAVY) AND REMARKS
    SERVICE (NAVY) AND REMARKS ()

Extract records where SERVICE=NAVY and REMARKS is blank

```
     SERVICE (NAVY) AND NOT REMARKS
     SERVICE (NAVY) AND REMARKS(     )
```

(4) Family Search

A family is a field of data broken down into equal units, usually containing a list of related items of information. Whenever a user specifies the name of a family in his request, the system will automatically search all units included in the family. In addition to addressing a whole group of units by a single family identifier, the user may also address a particular unit in the family.

Example:

A family name HOBBIES refers to a list of elements named HOB1, HOB2, and HOB3. A user may specifically refer to HOB1, HOB2, and HOB3. A reference to HOBBIES would address all three elements individually.

(5) Special Logical Connectors

Two special logical connectors, AND/REC and OR/REC, must be used in queries which have criteria on different record formats of a multi-format file. If these operators are not used then all the search criteria must be satisfied in a single record. However, there is no information in the user manual about the precedence relationships between these operators and the remaining TILE logical operators, except for the following two examples, where FORM indicates the format identifier field:

```
FORM(AA) AND ORG(G612) AND/REC FORM(BB) AND DOV>(740331)
FORM(AA) AND ORG(G612) OR/REC FORM(BB) AND DOV>(740331)
```

(6) Special Logical Operators

The keyword NOT may be used as a binary logical operator in place of AND NOT. The words IF and IS are null-words, having no effect on the query.

Three special logical operators are available: UNLESS, ALSO, and ANDEITHER. These operators may be used to factor logical expressions, similar to the way ordinary languages permit parentheses to be used to factor logical expressions to achieve more efficient query processing. These operators have not been included in the formal syntax for TILE in Figure A-2.

The UNLESS logical operator (synonym UNLS) defines an exception to the condition immediately preceding it. The

exception condition follows the UNLESS operator and is terminated by a comma or the end of the statement.

Factored Format	Basic Format
A UNLESS B	A AND NOT B
NOT A UNLESS B	NOT A OR B
NOT A UNLESS B OR C	NOT A OR B OR C
NOT A UNLESS B, OR C	NOT A OR B OR C

The ALSO group follows the word ALSO and is terminated by a comma or the end of a statement. When a statement containing the ALSO logical operator is expanded into the basic format, the ALSO group is conjuncted (i.e., AND) with all groups preceding the ALSO.

Factored Format	Basic Format
A OR B ALSO C	A AND C OR B AND C
A OR B ALSO C, OR D AND E	A AND C OR B AND C OR D AND E
A OR B ALSO C AND D	A AND C AND D OR B AND C AND D

The ANEITHER connector is similar to ALSO, except that it conjuncts (i.e., AND) the conditions preceding it with all groups that follow it.

Factored Format	Basic Format
A ANDEITHER B OR C	A AND B OR A AND C
A AND B ANDEITHER C AND D OR E	A AND B AND C AND D OR A AND B AND E

4. SEA WATCH QUERY FACILITY

a. Query Submission Protocol

The primary means of retrieving data from the Sea Watch System is through the Query function. The Query request consists, basically, of a command, output device, list of data elements to be output, a set of qualifiers and a sort element list if the optional sort capability is desired. The Query function can operate in either batch or interactive mode. The Sea Watch System has the capability to store queries for later use.

b. Data Structures

The Sea Watch data base is made up of a large number of files. The files are actually composed of one or more tables. Each table is a one-dimensional array of one or more items per table. The files are interrelated via pointers from one file to the other. The data within one file contains pointers to related data in other files. These pointers are actually elements in the file indicating the address location of related information in another file.

c. Retrieval Verbs

Data is retrieved from the Sea Watch data base using the Query function. A Query request consists of a command, an output device, a list of data elements to be output, a set of qualifiers to specify retrieval criteria, an optional list of data elements to be used as sort keys, and an optional file specification.

For the file specification (Figure A-3) the default is NMF. The file names N, M, F, and U correspond to Naval, Merchant, Fishing, and Unresolved respectively. The file specifications may indicate C or H for Current data or History data respectively. The default is to search both.

d. Syntax Description

The following comments supplement the formal syntax presented in Figure A-3. The syntax description does not include the output specification parameters. The maximum number of levels of parentheses is three (3). An element-name may be either a short name, a long name, or a data element number (D-number).

The BETWEEN operator is required with data elements representing latitude or longitude values, but may also be used with any other data elements. When BETWEEN is used with data elements representing name values, the two values must be separated by a semi-colon. The semi-colon is optional between values of other types (i.e., number, date, time, Lat/Long).

A blank value is represented by a blank between matching brackets ([ ]).

A date value in a set of criteria may be followed by a data qualifier (LAST, LASTA, LASTD, LASTV, LASTM), provided the element in the criteria is a date element or a date-time (DTG) element, and provided that the criterion occurs as the last criterion in the query. LAST (synonym is CURRENT) is used to retrieve a ship at its last known position. LASTA is used to retrieve a ship indicating its last position based upon some other qualities. LASTD, LASTV, LASTM are used to retrieve the last (most recent) deployment, voyage or movement respectively based upon additional qualifiers.

The logical "OR" is limited to a maximum of 23 uses per query.

```
<command>          ::=    print-phrase WHERE QUERY
                        | print-phrase WHERE <select-expr> [<file>]

<file>             ::=    FILE - <file-list> [C | H]

<file-list>        ::=    <file-name> [<file-list>]

<file-name>        ::=    N | M | F | U

<select-expr>      ::=    <and-expr> [OR <select-expr>]

<and-expr>         ::=    <paren-expr> [AND <and-expr>]

<paren-expr>       ::=    <criteria>
                        | (<select-expr>)

<criteria>         ::=    <element> <relation> <value-list>
                        | <element> FROM date TO date
                        | <element> BEFORE date
                        | <element> AFTER date
                        | <element> BETWEEN <value> <value>

<relation>         ::=    EQ | NQ | GR | GQ | LS | LQ | BETWEEN

<value-list>       ::=    <value> [; <value-list>]

<value>            ::=    [[*] string []*]
                        | date <date-qualifier>

<element>          ::=    element-name

<date-qualifier>   ::=    LAST[A | D | V | M]
```

Figure A-3. Syntax Description of Sea Watch Query Function.

*Note: Brackets ([]) must be used as value string delimiters if the value string contains any special characters or reserved words.

APPENDIX B
TIIN VALIDATION EXERCISES

1. INTRODUCTION

The TIIN Validation Exercises will illustrate different stages of processing in the TIIN System. This appendix will demonstrate the capabilities of the TIIN functional design by presenting a step-by-step translation of a query into the DIAOLS language. The following phases of TIIN processing will be featured:

- User query
- DRIF
- QNF
- QTF
- QTF Processor
- HL Generator
- HL Processor

2. USER QUERY

The user/analyst inputs a query in his local query language. The sample query in Figure B-1 is meant to suggest a typical query that might be entered into the TIIN System. The sample query does not represent an actual query language, and is presented for illustration purposes only.

```
STARTQ;
RETRIEVE (SHIPLENGTH > 200
     OR SHIPCOUNTRY = PO AND SHIPNAME = DAR POMORZA)
     AND OUTSIDE ROUTE (2, 4, 413000N, 735000W, 413000N, 713000W)
     AND INSIDE CIRCLE (100, 413000N, 724000W);
SHOW SHIPNAME, SHIPCOUNTRY;
ENDQ;
```

Figure B-1. Sample Input Query.

The sample query begins and ends with the keywords STARTQ and ENDQ. The statements in the query language are separted by semicolons. The verbs in the example are RETRIEVE and SHOW. The RETRIEVE verb selects records that satisfy relational conditions. The SHOW verb specifies which element values are to be retrieved from the selected records.

The intent of the query is to print out the name and country of origin of all ships which are either longer than 200 feet or Polish ships named "DAR POMORZA". The ships selected must be located outside the specified route, but inside the specified circle.

3. DRIF

The DRIF data structure is an interface between the TIIN/TAP and the TIIN/QIP. At this point in the processing the verbs and operators in the sample query have been translated into the TIIN standard verbs and operators. The DRIF has the block layout shown in Figure III-2. The host data base name and file name remain unspecified.

The Conditional table in the DRIF generated from the sample query is shown in Figure B-2. The entry immediately preceding the ROUTE operator in the Conditional table has been created by the DRIF Generator for the benefit of the QTF Generator and contains the number of operands associated with this instance of the ROUTE operator.

4. QNF

The QNF data structure is the output of the QNF Generator. In the QNF Generator the element names have been translated from local to TIIN Standard. The values have been converted to standard formats. The name of the file to be queried and the name and location of the data base containing the file have been determined.

Figure B-3 shows the conditional table of the QNF.

The file selected by the QNF Generator is the TALL-SHIPS File (found In Appendix E). This file is presumed to be located at the DIAOLS site. An entry containing the file name has been added to the conditional table.

5. QTF

The QTF data structure is the output of the QTF Generator. The QTF contains direct links between operators and their operands and is in a format which facilitates further processing. The contents of the QTF is equivalent to the contents of the QNF, only the format is different.

The symbolic format of the QTF of the sample query is shown in Figure B-4. The format of the QTF internal to a computer is described in Section III-4, but will not be used here.

Byte	188		
0	188		
	Byte Count	Type	String
2	12	3	SHIPLENGTH
14	5	6	200
19	3	2	GT*
22	13	3	SHIPCOUNTRY
35	4	6	PO
39	3	2	EQ*
42	10	3	SHIPNAME
52	13	6	DAR POMORZA
65	3	2	EQ*
68	3	2	AND*
71	3	2	OR*
74	3	6	2
77	3	6	4
80	9	6	413000N
89	10	6	0735000W
99	9	6	413000N
108	10	6	0713000W
118	3	6	6
121	3	2	ROUTE*
124	3	2	NOT*
127	5	6	100
132	9	6	413000N
141	10	6	0724000W
151	3	2	CIRCLE*
154	3	2	AND*
157	3	2	AND*
160	3	1	SELECT*
163	10	3	SHIPNAME
173	13	3	SHIPCOUNTRY
186	3	1	PRINT*

Figure B-2. DRIF Conditional Table of Sample Query.

*Indicates a one-byte type code denoted by a mnemonic from Figure III-5.

Byte	Byte Count	Type	String
0	184		
	Byte Count	Type	String
2	8	4	SHLENG
10	5	7	200
15	3	2	GT*
18	9	4	SHCOUNT
27	4	7	PO
31	3	2	EQ*
34	8	4	SHNAME
42	13	7	DAR POMORZA
55	3	2	EQ*
58	3	2	AND*
61	3	2	OR*
64	3	7	2
67	3	7	4
70	9	7	413000N
79	10	7	0735000W
89	9	7	413000N
98	10	7	0713000W
108	3	7	6
111	3	2	ROUTE*
114	3	2	NOT*
117	5	7	100
122	9	7	413000N
131	10	7	0724000W
141	3	2	CIRCLE*
144	3	2	AND*
147	3	2	AND*
150	3	1	SELECT*
153	8	4	SHNAME
161	9	4	SHCOUNT
170	3	1	PRINT*
173	12	9	TALL-SHIPS

Figure B-3. QNF Conditional Table of Sample Query.

*Indicates a one-byte code denoted by a mnemonic from Figure III-5.

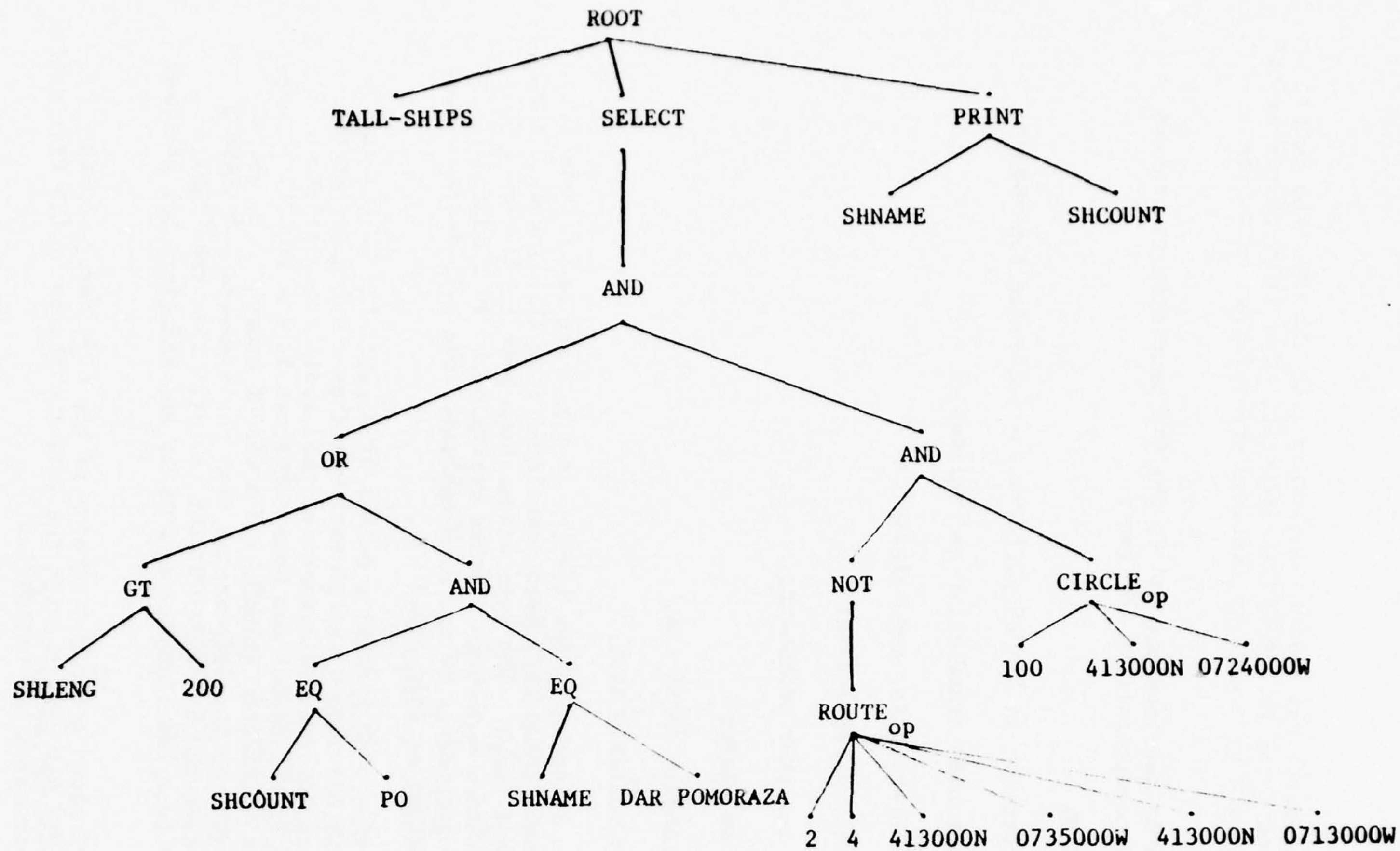


Figure B-4. QTF of Sample Query Showing Standard Network Names.

The QTF Generator has deleted one entry from the QNF, the entry containing the operand count for the ROUTE operator. The entry is not needed in the QTF since each QTF node already contains the information on the number of operands.

The ROOT node has been created by the QTF Generator to contain pointers to all major components of the query.

6. QTF PROCESSOR

The QTF Processor in DIAOLS performs the following functions:

- o Element name translation and validation
- o Value translation and validation
- o Operator existence test
- o WITH operator processing
- o Verb selection
- o Parentheses depth check
- o Query validity test.

Figure B-5 illustrates the results of the first four of the above functions. The element names have been translated from TIIN standard into the local host element names. The data values have been translated and validated. The operators have been checked for validity in DIAOLS. If the query contained any occurences of the WITH operator, the WITH operator would also have been processed at this stage.

The verb selection process consists of mapping the SELECT and PRINT branches of the QTF into a set of verb branches which exist at the host DBMS. The result of verb selection for the sample query is shown in Figure B-6. The SELECT branch has been converted into a REQUEST branch, a TEXTSCAN branch, a GEOGRAPHIC branch, and a CIRCLE branch. The above choices were made based on the information that the elements SHIP-LENGTH and SHIP-NAME are inverted, but SHIP-COUNTRY is not. The verb PRINT translates directly into the DIAOLS verb COMPOSE and therefore has not been changed.

Figure B-7 shows the final version of the QTF, ready for the HL Generator. The QTF has gone through the last two phases of QTF Processing, the parentheses depth check and validation.

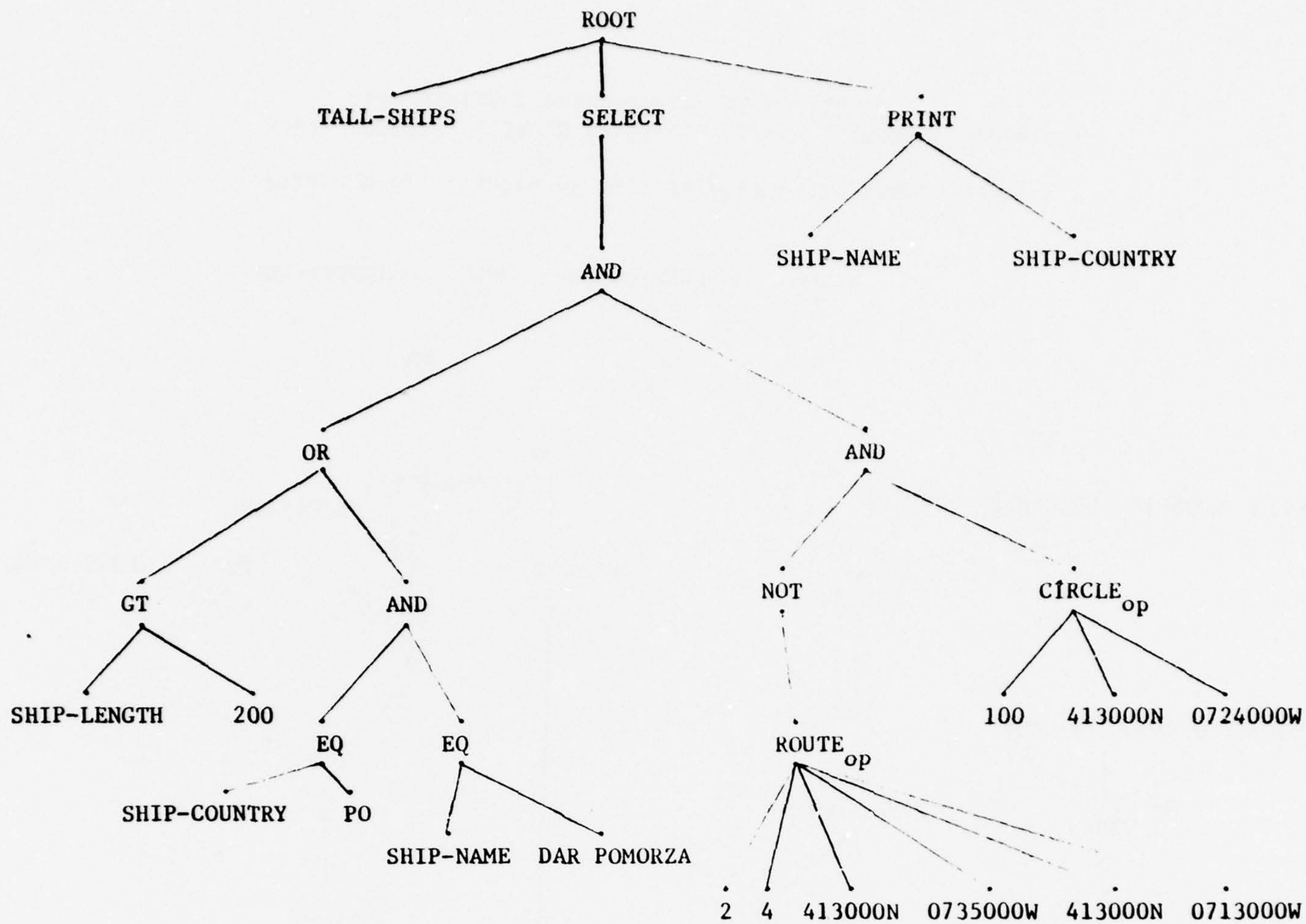


Figure B-5. QTF of Sample Query Showing Host Element Names and Values.

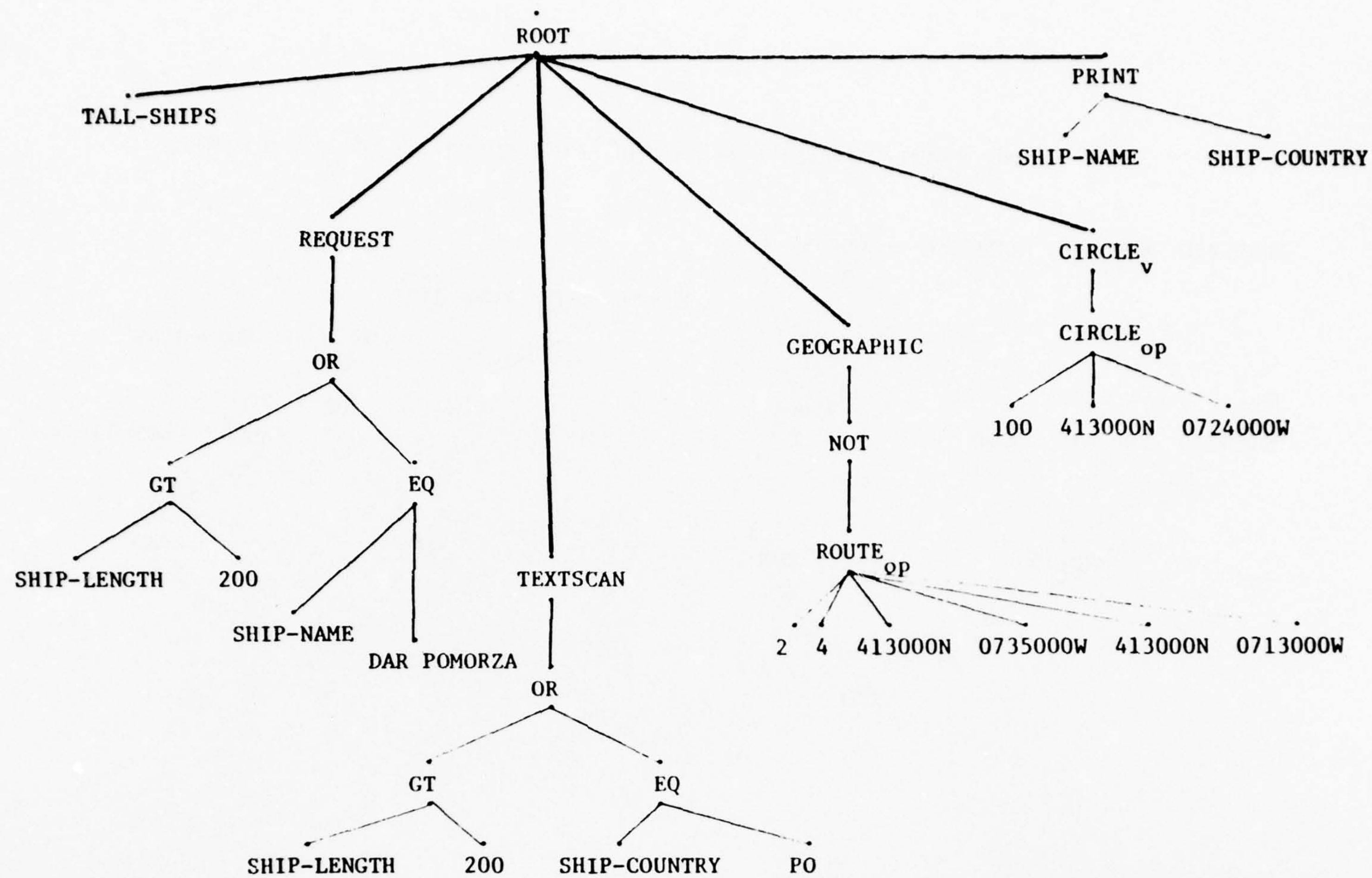


Figure B-6. Results of Verb Selection for Sample Query.

Note: CIRCLE$_v$ (CIRCLE verb) and CIRCLE$_{op}$ (CIRCLE operator) are distinct nodes in the above figure.

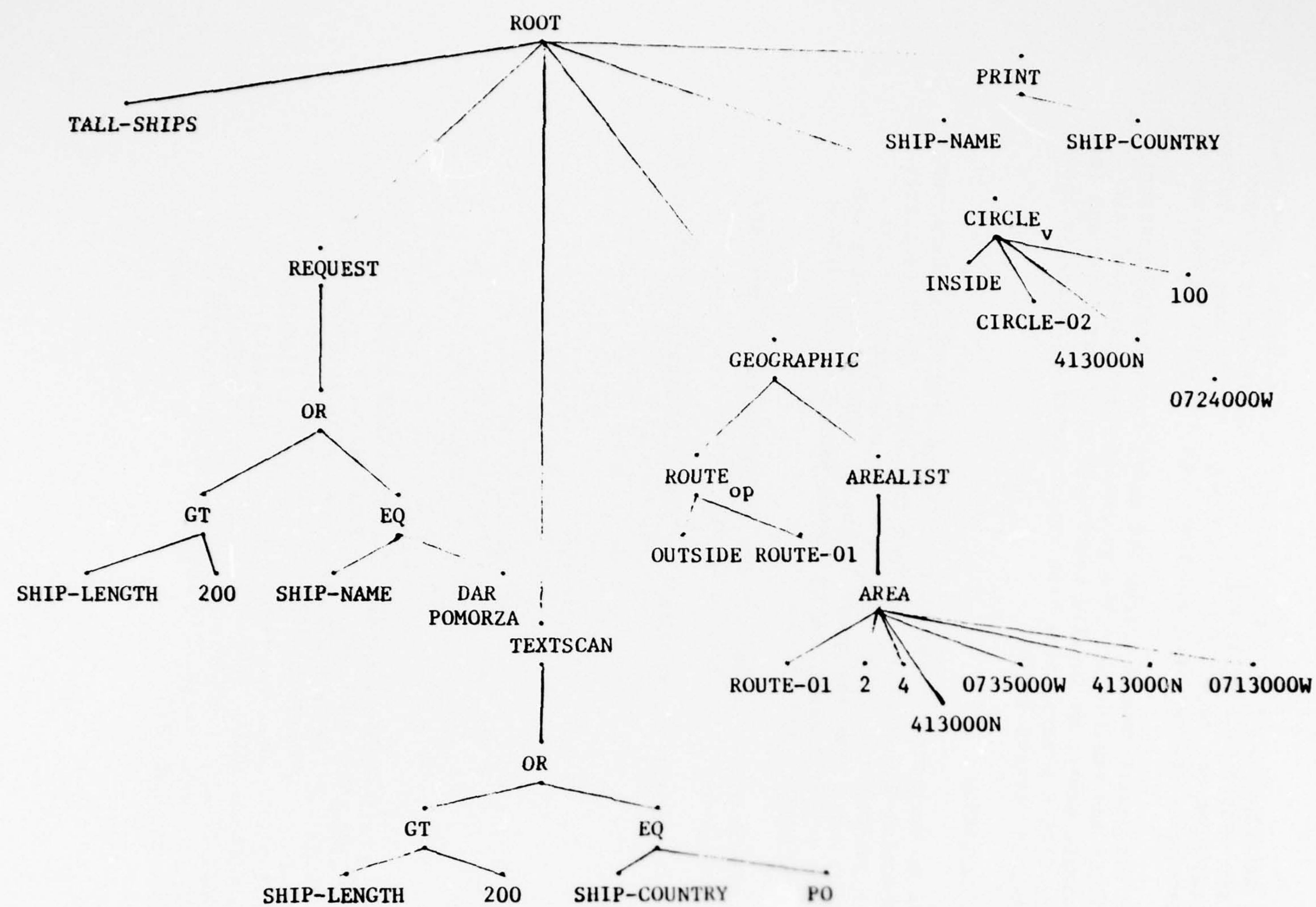


Figure B-7. Sample Query at the End of QTF Processor Phase.

The parentheses depth-check module uses the operator precedence specified in the syntax table (see Figure IV-24) to determine how many levels of parentheses are required by the query. In the case of the sample query, no parentheses are required and therefore the query passes the test.

The validation module checks the query for certain constructs invalid in DIAOLS and restructures the geographic operators. In the case of the sample query, no invalid constructs have been found and the GEOGRAPHIC and CIRCLE phrases have been restructured according to the rules described in Section IV-7c.

7. HL GENERATOR

The HL Generator processes the QTF and produces a character string representing the query in the host language. The HL Generator consists of a generalized driver program which references a table of expansion rules for a specific query language. The expansion rules describe to the driver program how to expand each tree node into a host language character string.

A DIAOLS syntax table with the expansion rules necessary to process the sample query is shown in Figure IV-24.

The character string output of the HL Generator is shown in Figure B-8.

```
FILE,␣TALL-SHIPS␣REQUEST␣IF␣SHIP-LENGTH␣GT␣
:200:␣OR␣SHIP-NAME␣EQ␣:DAR POMORZA:.␣
TEXTSCAN␣IF␣SHIP-LENGTH␣GT␣:200:␣OR␣SHIP-COUNTRY␣
EQ␣:PO:.␣GEOGRAPHIC,␣NEW␣IF␣OUTSIDE␣ROUTE-01.␣
ROUTE-01␣EQ␣2,␣4,␣413000N,␣0735000W,␣413000N,␣
0713000W␣CIRCLE,␣NEW␣IF␣INSIDE␣CIRCLE-02␣
CIRCLE-02␣EQ␣413000N,␣0724000W,␣100␣COMPOSE,␣
NEW,␣1␣DATA␣=␣SELECT␣SHIP-NAME␣SHIP-COUNTRY.␣
END.␣DISPLAY␣BYE.
```

Figure B-8. HL Generator Output of Sample Query

8. HL PROCESSOR

The HL Processor converts the character string output of the HL Generator into a syntactically valid query in the host query language. The HL Processor performs line length checks, line number and end-of-line character insertion, and any additional processing.

Figure B-9 shows the completed DIAOLS query.

```
005 FILE,␣TALL-SHIPS
010 REQUEST
015 IF␣SHIP-LENGTH␣GT␣:200:
020 OR␣SHIP-NAME␣EQ␣:DAR POMORZA:.
025 TEXTSCAN
030 IF␣SHIP-LENGTH␣GT␣:200:
035 OR␣SHIP-COUNTRY␣EQ␣:PO:.
040 GEOGRAPHIC,␣NEW
045 IF␣OUTSIDE␣ROUTE-01.
050 ROUTE-01␣EQ␣2,␣4,␣413000N,␣0735000W,
055 413000N,␣0713000W
060 CIRCLE,␣NEW
065 IF␣INSIDE␣CIRCLE-02
070 CIRCLE-02␣EQ␣413000N,␣0724000W,␣100
075 COMPOSE,␣NEW,␣1␣DATA␣=␣SELECT
080 SHIP-NAME␣SHIP-COUNTRY.
085 END.
090 DISPLAY
095 BYE.
```

Figure B-9. Completed DIAOLS Query

APPENDIX C

BIBLIOGRAPHY

1. DATA BASES

Abraham, C.T., et al, "Formatted File Organization Techniques," IBM Corp., RADC-TR-67-329, June 1967, AD 819 338L.

Aschim, Frode, "Data Base Networks - An Overview," Management Informatics, (1974) Vol. 3, No. 1.

Astrahan, M.M., and Chamberlin, D.D., "Implementation of a Structured English Query Language," Communications of the ACM, Oct. 1975, Vol. 18, No. 10, pp. 580-588.

Bleier, Robert E., "Treating Hierarchical Data Structures in the SDC Time-Shared Data Management System (TDMS)," 1967 ACM National Meeting, pp. 41-49.

Booth, Grayce M., Honeywell Information Systems, Phoenix, Arizona, "The Use of Distributed Data Bases in Information Networks," First International Conference on Computer Communication: Impacts and Implications, Oct. 24-26, 1972.

Boyce, R.F., Chamberlin, D.D., King III, W.F., and Hammer, M.M., "Specifying Queries as Relational Expressions: SQUARE," IBM Technical Report RJ 1291, (Oct. 1973).

Canaday, R.H., et al, "A Back-end Computer for Data Base Management," October 1974, Communications of the ACM, Vol. 17, No. 10.

Chamberlin, D.D. and Boyce, R.F., "SEQUEL: A Structured English Query Language," Proc. 1974 ACM SIGFIDET Workshop, Ann Arbor, Michigan, (April 1974), pp. 249-264.

Chandra, A.N., "Some Considerations in the Design of Homogeneous Distributed Data Bases," IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

CODASYL Development Committee, "An Information Algebra," Communications of the ACM, Vol. 4, (April 1962), pp. 190-204.

CODASYL Systems Committee, "Introduction to 'Feature Analysis of Generalized Data Base Management Systems,'" Communications of the ACM, Vol. 14, No. 5 (May 1971), pp. 308-318.

CODASYL Systems Committee, "Feature Analysis of Generalized Data Base Management Systems," New York, May 1971.

CODASYL Data Base Task Group Report, ACM, April 1971.

Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, Vol. 13, No. 6, (June 1970), pp.377-387.

____, "Seven Steps to Rendezvous with the Casual User," IFIP TC-2 Working Conference on Data Base Management Systems, Cargese, Corsica, 1-5 April 1974.

Computing Surveys, ACM, Vol. 8, No. 1, March 1976.

Date, C.J., "An Architecture for High-Level Language Data Base Extensions," 1976 SIGMOD Conference.

____, and Hopewell, P., "File Definition and Logical Data Independence," Proceedings of the 1971 ACM SIGFIDET Workshop.

Defense Intelligence Agency Regulation 65, Intelligence Information Systems, Defense Intelligence Data Standards System (DIDSS).

Earley, Jay "Toward an Understanding of Data Structures," Comm. of the ACM, Vol. 14, No. 10, Oct. 1971.

Fry, J.P., Smith, D.P., and Taylor, R.W., "An Approach to Stored Data Definition and Translation," Proc. ACM SIGFIDET Workshop on Data Definition and Access, Denver, Colo., 1972, pp. 13-55.

____, "Stored Data Definition and Translation Approach to the Data Portability Problem," Data Translation Project Report, University of Michigan, Ann Arbor, Mich., Feb. 1974.

Ghosh, S.P., and Senko, M.E., "String Path Search Procedures for Data Base Systems," IBM Journal of Research and Development, Vol. 18, No. 5, (Sept. 1974), pp. 408-422.

Jervis, B., Parker, J., "An Approach for a Working Relational Data System," Dept. of Computer Science, University of British Columbia, Vancouver, Canada.

Kellogg, Charles H., " A Natural Language Compiler for On-Line Data Management," AFIPS, (1968), FJCC.

Logicon, Inc., Final Report - Study of the Multi-Language Problem in COINS, Vol. 1. - Recommended Solutions, Vol. 2. - COINS DBMS Feature Analysis, (May 1975) Logicon San Diego, CA.

Lum, V.Y., Shu, N.C., Housel, B.C., "Data Translation, Par I: A General Methodology for Data Conversion and Restructuring," IBM (San Jose), RJ 1525 (#23112), July 1975.

Marcus, Richard S., "A Translating Computer Interface for a Network of Heterogeneous Interactive Information Retrieval Systems," MIT, Electronic Systems Laboratory, 1972.

Merten, A.G., and Fry, J.P., "A Data Description Language Approach to File Translation," ACM Proc. SIGMOD Workshop on Data Description, Access and Control, Ann Arbor, Mich., (1974), pp. 191-205.

Plagman, G.K., and Altshuler, G.P., "A Data Dictionary/Directory System Within the Context of an Integrated Corporate Data Base." AFIPS, 1972, FJCC.

Schneider, L.S., "A Relational View of the DIAM," Proceedings 1976 ACM SIGMOD International Conference on Management of Data, Washington, D.C., (June 1976), pp. 75-90.

____, Spath, C.R., "Quantitative Data Description," Proceedings 1975 ACM SIGMOD International Conference on Management of Data, San Jose, California, (May 1975), pp. 167-185.

Senko, M.E., Altman, E.B., Astrahan, M.M., and Fehder, P.L., "Data Structures and Accessing in Data-Base Systems," IBM Systems Journal, Vol. 12, No. 1, (1973), pp. 30-93.

Shoshani, Ari, "Data Sharing in Computer Networks," 1972 Wescon Tech. Papers, System Development Corp.

____, and Spiegler, I., "The Integration of Data Management Systems on a Computer Network," American Institute of Aeronautics and Astronautics, Computer Network Systems Conference, Huntsville, Ala., April 1973.

____, Brandon, K., "The Implementation of a Logical Data Base Converter," October 1975, System Development Corporation, TM-5590/000/00.

Shu, N.C., Housel, B.C., and Lum, V.Y., "Data Translation, Part III. CONVERT: A High Level Translation Definition Language for Data Conversion." IBM Res. Rep. RJ 1515, Feb. 1975.

Sibley, E.H., and Taylor, R.W., "A Data Definition and Mapping Language," Comm. ACM, Vol. 16, No. 12, (Dec. 1973), pp. 750-759.

Smith, J.M., Chang, P., "Optimizing the Performance of a Relational Algebra Data Base Interface," Communications of the ACM, Vol. 18, No. 10., Oct. 1975.

Stamen, Jeffrey P., Shuford, Dorothy, "Beginner's Manual for the JANUS Prototype System," The Cambridge Project, M.I.T, (Sept 1972)

_____, Wallace, Robert M., "JANUS: A Data Management and Analysis System for the Behavioral Sciences," Proceedings of the 1973 Annual Conference of the ACM, Atlanta, Georgia (August 1973).

Uhrowczik, P.P., "Data Dictionary/Directories." IBM Systems Journal, Vol. 12, No. 4 (1973).

Wong, E., Youssefi, K., "Decomposition - A Strategy for Query Processing," ACM Transactions on Data Base Systems, Vol. 1, No. 3, (Sept 1976), pp. 223-241.

2. INCO PUBLICATIONS

Development Planning Factors for a Transparent Integrated Intelligence Network, INCO, INC., 1974.

Technical Proposal for a Transparent Integrated Intelligence Network, INCO, INC., 5 March 1975.

The Transparent Intelligence Language Facility, Final Technical Report, INCO, INC., McLean, Virginia, June 1976.

The Transparent Integrated Intelligence Network Development Plan, INCO, INC., McLean, Virginia, 15 November 1975.

Russek, Marianne, Distributed Data Base Information Systems Technology, INCO, INC., McLean, Virginia, 1975.

3. LANGUAGE MANUALS

DIAOLS ADP User's Guide for On-Line and Remote/Local Batch Operations, DIA, Washington, D.C., 1974.

On-Line System Support Center (OSSC) User's Guide to DIAOLS/COINS (To be published).

SEAWATCH External User's Manual, NOSIC, 1974.

TIPS Interrogation Language (TILE) Training Manual for NSA TIPS and COINS Users, NSA, 1972.

TOSS Information Management System (TIMS) On-Line User's Manual, July 1975, INCO, INC., McLean, Virginia.

APPENDIX D

GLOSSARY

COINS - The Community On-Line Intelligence Network Subsystem provides batch oriented access to intelligence data bases at NSA, DIA, CIA and NMIC.

Conditional Table - The conditional table is the part of the DRIF and QNF structures which contains the record selection criteria and the list of elements to be retrieved.

DBDL - The Data Base Description Language is a TILF-supported language used to describe the contents and structure of network data bases.

DBMS - A Data Base Management System manages and accesses a data base and provides responses to queries received from users.

DIAOLS - The Defense Intelligence Agency On-line System is a host DBMS on the COINS network.

DRIF - Data Request Intermediate Format is a data structure used to pass queries from a user (e.g., TAP) to QIP at the user node. The DRIF is query language independent.

Explicit Element - A data field existing in a data base, also referred to as an element; the opposite of Implicit Element.

HL - The Host Language is the query language in which a query is expressed to retrieve data from a host DBMS serving the information needs of the intelligence network users.

HL Generator - This module of the host QIP accepts the validated query in tree format (QTF) and converts it into the query language of the host DBMS.

HL Processor - This module of the host QIP accepts from the HL Generator a character string representing a query in the host language and performs any final adjustments required before submitting the query to the host DBMS.

Host (DBMS) - In this report, a host always refers to a data base and its associated DBMS accessible via a communications network.

Host NAD - A part of the NAD containing information used by the Host QIP at the host node to translate network standard format (QNF) queries into a host query language.

Host Node - In this report a host node is a network node which provides access to one or more data bases of an integrated intelligence network. The TIIN software at a host node will receive a QNF version of a user query and will perform operations on it to make it acceptable to the host data base management system. The DBMS provides a response to the query, which is sent to the TIIN software at the host node for normalization, preparatory to returning the query result to the user node on the network.

Host QIP - The Query Intermediate Processor at the host node validates a query given the restrictions imposed by the host query language and data base, and translates the query into the host query language.

Host RN - Those modules of the RN subsystem dealing with DBMS response processing at the host node. The function of the Host RN is to normalize DBMS responses into a TIIN standard response format, RNF.

Implicit Element - A data element whose value is global to all records in a file and thus is associated with the file name rather than being explicitly represented in the file.

NAD - The Network Access Directory is a distributed data base containing information about intelligence data bases in the network. Information contained in the NAD allows for element name translation, value translation, and host selection during TIIN processing.

Network - A network is a complex consisting of two or more interconnected computers.

NMIC - National Military Intelligence Center, Arlington, Virginia.

Node - A data processing site with its own communications interface to an extended, integrated network. In this report it is assumed that a node will have a PDP-11 with TOSS software, including TIIN with a Network Access Directory.

QDNC - The Query Distribution and Network Communications subsystem is a proposed TIIN subsystem concerned with the distribution of a query into the extended network and the tracking of queries distributed and responses received. It interfaces with the TOSS communications handler (TISS) which handles all message traffic to and from a node.

QIP - The Query Intermediate Processor is the collection of TIIN modules which transform a user data request into a standard query language and data base independent format for transmission to host nodes, and subsequently into a format compatible with a host node's data base management system.

QNF - Query Normal Format is a compact data structure used to transfer queries from the user node to the host node.

QNF Generator - This module of the User QIP produces the QNF version of a user query from the DRIF version.

QTF - The Query Tree Format is a data structure used by the host QIP at the host node, to allow easy validation and restructuring of the query.

QTF Generator - This module of the host QIP produces the QTF version of a query from the QNF version.

QTF Processor - This module of the host QIP validates and restructures the query, and performs element name and data value translation from network standard to host DBMS names and values.

RADC - Rome Air Development Center, Griffiss Air Force Base, Rome, New York.

RN - Response Normalization is a proposed TIIN subsystem which will receive the response to a user query from the host DBMS, convert it into TIIN standard format (RNF) for transmission to the user node, merge all standardized responses resulting from a single user request (DRIF), and present the user with a finished report.

RNF - The Response Normal Format is a TIIN standard data structure, used to transmit a query response over a network.

RPN - The Reverse Polish Notation is a representation technique in which operand expressions occur in a linear sequence immediately before the associated operator.

SSB - The Standard Software Base is a collective phrase encompassing TOSS-related and other software which provides a baseline for Air Force network applications.

TAP - The Terminal Access Procedures is a TIIN subsystem which allows a user to make a data request and to gain information about data base content and structure. TAP includes a language translator to produce the DRIF version of the user query to interface with the user QIP.

TEC- TOSS Exchange Center provides network communications control for bulk data and conversational message traffic.

TEL - The Transparency Examples Language is a hypothetical query language used to illustrate transparency features relative to a host query language.

TIIN - Transparent Integrated Intelligence Network.

TILE - TIPS Interrogation Language.

TILF - The Transparent Intelligence Language Facility is a TIIN subsystem concerned with creation and maintenance of the Network Access Directory (NAD).

TIMS - TOSS Information Management System is a data management system which gives the analyst on-line capabilities for file development, network file access, and language processing.

TIPS - Technical Information Processing System, a DBMS at NSA.

TISS - Terminal Independent Support System is a set of systems software and application programs that provides network capabilities to users/analysts through on-line terminals connected to minicomputers.

TOSS - The Terminal Oriented Support System is a multi-faceted information management and handling system which aids in integrating distributed intelligence resources in a cohesive, world-wide network. It is a hardware/software system employing minicomputers as stand-alone, terminal-oriented processors connected to local host computers to form nodes, which are in turn interconnected by communication links to form a widespread network. Components of TOSS include TEC, TIMS, TISS, TTDL.

TTDL - Terminal Transparent Display Language is a transparent terminal handling subsystem of TOSS.

User NAD - A part of the NAD containing information used by the User QIP at the user node to translate user element names and values into TIIN standard element names and values.

User node - The user node is a network node which provides terminal support through which an analyst initiates a query. A user node may have an associated DBMS. The TIIN software at the user node is where the user query is initially processed for distribution to host nodes.

User QIP - The Query Intermediate Processor at the user node consists of a single module, the QNF Generator. The functions of the User QIP include element name translation, data value translation, and host selection.

User RN - Those modules of the RN subsystem dealing with DBMS response processing at the user node. The function of the User RN is to merge all standardized responses resulting from a single user request (DRIF) and to generate a report in a format requested by the user.

WICS - Worldwide Intelligence Communications System.

WCF - WICS Common Format.

APPENDIX E

TALL-SHIPS FILE

1. INTRODUCTION

A sample file has been developed to aid in the understanding of the TIIN System. This file is named "TALL-SHIPS" and contains data concerning a fleet of sailing ships. The examples in the validation exercises of this report are taken from the Tall-Ships File.

2. LOGICAL FILE STRUCTURE

The logical file structure of the Tall-Ships File is compatible with the structure of files and records in DIAOLS/COINS. A file is a collection of related records treated as a unit. A record is a collection of related items of data treated as a unit. The record layout for the Tall-Ships File is shown in Figure E-1. All of the significant parts of the record are shown and the relationships to each other established.

Relational periodics tie or relate data in one element to data in one or more other elements by means of like subscripts associated with each of those elements. In the Tall-Ships record layout, PORT, DEPARTURE-DATE and ARRIVAL-DATE are the elements that contain relational periodics. Note that each of these elements has a maximum of five fields of recurring values. Note also that each instance of a relational periodic element has a subscript shown in parentheses. The subscript is stored as the first two characters of each field for each value of a relational periodic element, and is used to "relate" a value of one relational periodic element to the value of any other relational periodic element with the same subscript.

The data contained in Tall-Ships File is shown in Figure E-2.

FORMAT SIZE	SAMPLE VALUE	ELEMENT NAME
8 A/N	76Ø7Ø7	DATE-OF-CHANGE
4 A/N	TSØ7	RECORD-ID
20 A/N	MIRCEA	SHIP-NAME
7 A/N	2Ø4816N	LATITUDE-DEG-MIN
8 A/N	1Ø64723E	LONGITUDE-DEG-MIN
2 A	US	COUNTRY-CODE
10 A	ACTIVE	STATUS
3 N	2Ø4	SHIP-LENGTH
20 A/N	(Ø1)NEW YORK	PORT
20 A/N	(Ø2)BALTIMORE	
20 A/N		
20 A/N		
20 A/N		
8 N	(Ø1)76Ø7Ø3	ARRIVAL-DATE
8 N	(Ø2)76Ø71Ø	
8 N		
8 N		
8 N		
8 N	(Ø1)76Ø7Ø8	DEPARTURE-DATE
8 N	(Ø2)76Ø714	
8 N		
8 N		
8 N		

Figure E-1. Record Layout For Tall-Ships File.

```
RECORD-ID            =TSØ1         SHIP-LENGTH      =  295
DATE-OF-CHANGE       =76Ø628       PORT             =(Ø1) NEW YORK
SHIP-NAME            =US EAGLE                       (Ø2) BALTIMORE
LATITUDE-DEG-MIN     =400000N      ARRIVAL-DATE     =(Ø1) 76Ø7Ø1
LONGITUDE-DEG-MIN    =720000W                        (Ø2) 76Ø711
COUNTRY-CODE         =US           DEPARTURE-DATE   =(Ø1) 76Ø7Ø9
STATUS               =ACTIVE                         (Ø2) 76Ø714

RECORD-ID            =TSØ2         SHIP-LENGTH      =  282
DATE-OF-CHANGE       =76Ø7Ø7       PORT             =(Ø1) NEW YORK
SHIP-NAME            =GORCH FOCK                     (Ø2) BALTIMORE
LATITUDE-DEG-MIN     =403912N      ARRIVAL-DATE     =(Ø1) 76Ø7Ø2
LONGITUDE-DEG-MIN    =715855W                        (Ø2) 76Ø71Ø
COUNTRY-CODE         =WG           DEPARTURE-DATE   =(Ø1) 76Ø7Ø8
STATUS               =ACTIVE                         (Ø2) 76Ø717

RECORD-ID            =TSØ3         SHIP-LENGTH      =  269
DATE-OF-CHANGE       =76Ø7Ø9       PORT             =(Ø1) NEW YORK
SHIP-NAME            =MIRCEA                         (Ø2) BALTIMORE
LATITUDE-DEG-MIN     =384255N      ARRIVAL-DATE     =(Ø1) 76Ø63Ø
LONGITUDE-DEG-MIN    =742431W                        (Ø2) 76Ø711
COUNTRY-CODE         =RO           DEPARTURE-DATE   =(Ø1) 76Ø7Ø9
STATUS               =ACTIVE                         (Ø2) 76Ø716

RECORD-ID            =TSØ4         SHIP-LENGTH      =  353
DATE-OF-CHANGE       =76Ø7Ø2       PORT             =(Ø1) NEW YORK
SHIP-NAME            =ESMERALDA                      (Ø2) BALTIMORE
LATITUDE-DEG-MIN     =384739N      ARRIVAL-DATE     =(Ø1) 76Ø7Ø2
LONGITUDE-DEG-MIN    =7359Ø7W                        (Ø2) 76Ø712
COUNTRY-CODE         =CH           DEPARTURE-DATE   =(Ø1) 76Ø71Ø
STATUS               =ACTIVE                         (Ø2) 76Ø717

RECORD-ID            =TSØ5         SHIP-LENGTH      =  253
DATE-OF-CHANGE       =76Ø7Ø9       PORT             =(Ø1) NEW YORK
SHIP-NAME            =DANMARK                        (Ø2) BALITMORE
LATITUDE-DEG-MIN     =41Ø244N      ARRIVAL-DATE     =(Ø1) 76Ø7Ø3
LONGITUDE-DEG-MIN    =692949W                        (Ø2) 76Ø711
COUNTRY-CODE         =DK           DEPARTURE-DATE   =(Ø1) 76Ø7Ø6
STATUS               =ACTIVE                         (Ø2) 76Ø723

RECORD-ID            =TSØ6         SHIP-LENGTH      =  291
DATE-OF-CHANGE       =76Ø713       PORT             =(Ø1) NEW YORK
SHIP-NAME            =DAR POMORZA                    (Ø2) BALTIMORE
LATITUDE-DEG-MIN     =374818N      ARRIVAL-DATE     =(Ø1) 76Ø7Ø1
LONGITUDE-DEG-MIN    =75Ø1Ø3W                        (Ø2) 76Ø715
COUNTRY-CODE         =PO           DEPARTURE-DATE   =(Ø1) 76Ø713
STATUS               =ACTIVE                         (Ø2) 76Ø718
```

Figure E-2. Data Contained in the TALL-SHIPS File.

(Page 1 of 2).

```
RECORD-ID              =TSØ7          SHIP-LENGTH       =  115
DATE-OF-CHANGE         =76Ø7Ø8        PORT              =(Ø1) NEW YORK
SHIP-NAME              =EENDRACHT                        (Ø2) BALTIMORE
LATITUDE-DEG-MIN       =391457N       ARIVAL-DATE       =(Ø1) 76Ø7Ø2
LONGITUDE-DEG-MIN      =743923W                          (Ø2) 76Ø71Ø
COUNTRY-CODE           =NL            DEPARTURE-DATE    =(Ø1) 76Ø7Ø7
STATUS                 =ACTIVE                           (Ø2) 76Ø713

RECORD-ID              =TSØ8          SHIP-LENGTH       =  333
DATE-OF-CHANGE         =76Ø712        PORT              =(Ø1) NEW YORK
SHIP-NAME              =AMERIGO VESPUCCI                 (Ø2) BALTIMORE
LATITUDE-DEG-MIN       =375946N       ARRIVAL-DATE      =(Ø1) 76Ø7Ø3
LONGITUDE-DEG-MIN      =685139W                          (Ø2) 76Ø711
COUNTRY-CODE           =IT            DEPARTURE-DATE    =(Ø1) 76Ø7Ø9
STATUS                 =ACTIVE                           (Ø2) 76Ø719
```

Figure E-2. Data Contained in the TALL-SHIPS File.

(Page 2 of 2)

APPENDIX F

DEVELOPMENT BACKGROUND

1. TOSS BACKGROUND

Over the past four years, Terminal Oriented Support System (TOSS) concepts and implementation methodologies have undergone research and development by INCO, INC., of McLean, Virginia, for the Rome Air Development Center (RADC). The objective is to provide intelligence analysts with transparent access to computer files, remote data bases and other analysts world-wide through supporting communication facilities. TOSS development has proceeded along lines which seek to extend the concepts of transparency and distributed processing to all relevant aspects of the intelligence ADP environment.

The approach taken by INCO in implementing these concepts has been to build upon existing technology, such as that used in the NMIC Modernization project. Briefly, TOSS is a multifaceted information management and handling system which aids in integrating distributed intelligence resources in a cohesive, world-wide network. It is a hardware/software system employing minicomputers as stand-alone, terminal-oriented processors connected to local host computers to form nodes, which are in turn interconnected by communication links to form a widespread network. The principal components of TOSS include TOSS Exchange Center (TEC) providing network communications control for bulk data and conversational message traffic; Terminal Independent Support System (TISS) providing the capability for on-line analyst interaction through transparency of communication protocols; TOSS Information Management System (TIMS), a data management system providing on-line capabilities for network file access and for development and query of hierarchical files; and finally, the Terminal Transparent Display Language (TTDL) facilitating the development of applications programs and stressing the concepts of terminal transparency.

2. TIIN DEVELOPMENT

The TIIN development effort seeks to extend the transparency and distributed processing concepts of TOSS to include query language transparency and data base transparency. TIIN will be based on the hardware/software environment of TOSS with its access to a widespread network of data base management systems with different query languages, different technologies for file structure, different conventions for encoding data.

The eventual aim of the TIIN is to allow the user/analyst to converse with the network without regard for file structure and host dependent considerations. The user should feel as though he is accessing information available to him at his local node when, in fact, he is accessing remote data bases, perhaps world-wide.

The TIIN development follows an integrated, step-by step approach consisting of successive development stages. Each of the stages of TIIN development provides a discrete advance in user capabilities. Advances in system capability require corresponding technological advances. The development of a technology base for TIIN is planned to be systematic and cumulative; most technological features are applicable to all TIIN development stages and are enhanced at each level.

The user capabilities added at each TIIN development stage are described below:

a. Transparent Access to a Local Data Base

The analyst may obtain a description of entry to and use of a local minicomputer data base without specific knowledge of the data base management system.

b. Transparent Access to Distributed Homogeneous DBMS

Local and remote data bases having the same data management system appear as a single data base from the viewpoint of analyst accessibility. Capabilities such as stored queries and distributed standing queries are to be integrated into the network.

c. Transparent Access to Distributed Heterogeneous DBMS

Data base resources available to the analyst are extended to include data bases accessed by data management systems other than a single (common) DBMS, but which are still within the minicomputer network.

d. Integration of Host Computers and Foreign Netorks

The network available to the analyst is extended to include data bases in host systems and foreign networks which are connected to the minicomputer network via its nodes. The concept of a network user is extended to include analysts not connected to one of the host systems.

e. Integration of Work Stations Having Intelligent Terminals

Network capabilities are fully integrated with specially designed features and capabilities afforded by an intelligent terminal implemented as an analyst work station in the intelligence community.

3. GUIDING CONCEPTS

The development of the Transparent Integrated Intelligence Network has proceeded in accord with several major principles. They include transparency, data base integration, data sharing, and distributed processing. Each of these ideas will be dealt with in this subsection.

a. Transparency

This concept implies that detailed system knowledge is not required by the user to access and operate network elements. In a transparent environment, analysts can access information from unfamiliar data bases using procedures with which they are familiar. Query translation, data translation, and reconciliation are provided by special software modules, relieving analysts and network elements from requirements for mutual familiarity.

The notion of transparency, as applied to an information processing system, refers to those system characteristics which allow a user to ignore, indeed be unaware of, complexities and diversities which are not relevant to his purpose. For example, problem oriented languages such as COBOL and FORTRAN make machine language transparent to their users.

The concept of transparency as used in the TIIN effort is best communicated in terms of levels of transparency.

(1) No Transparency

Without transparency the analyst must use separate terminals independently connected to each external system. He must use the precise system access procedures as well as that query language associated with the data base he is accessing (the native language of the data base).

(2) Communications Transparency

At this level, the analyst would be able to retrieve data from external systems at his own analyst station, that is, communications and line protocols would be made transparent to the analyst, as well as logon procedures to the external systems. The analyst would still be required to use the data base access procedures (query languages) of the various external systems in order to retrieve data.

(3) Query Language Transparency

At this level, the analyst is able to retrieve data from diverse data bases and data files using a single query procedure from his own terminal. He does not have to know the query language on the particular external data base. There are, however, demands made on the analyst's time and knowledge inasmuch as he must be aware of the dispersion, structure and conventions for data bases being accessed.

(4) Data Base Transparency

At this level, dispersion of data resources among the various files and separate data bases is transparent to the analyst. Data is retrieved using one data access procedure from the analyst's terminal; computer software is used to separate the single query into separate queries to be directed at various data bases and files containing the requested information. In addition, the disparity among similar data element names and coded data field values is resolved.

b. Integrated Data Base

The intelligence community shares data requirements. The accuracy and completeness of an intelligence analysis presupposes that all relevant data be available to the analyst.

The concept of an integrated DBMS is concerned with providing all users with access to all data bases on a network. The concept presupposes the existence of a common data access language which is implemented system-wide. Integration may be achieved in many different ways: merge all data bases, provide a common DBMS with distributed data bases, or provide query language transparency and data base transparency for existing systems.

c. Delegated Production

The concept of delegated production seeks to minimize data maintenance costs by designating specific sites with non-overlapping responsibilities for data collection. A natural corrolary of designating specialized data production centers is that analysts must be able to access and cross correlate the data from the non-overlapping data bases. The volume of data updates and the size of the data bases involved make it non-cost-effective to maintain copies of the data bases at the data analysis sites separated from the data collection sites.

d. Distributed Processing

The concept of distributed processing seeks to allocate computer resources on a network basis so that response time and overall network utilization are optimized. Balanced availability of computer resources permits the community's computational requirements to be satisfied even when some processors become temporarily unavailable to the network.

4. THE ANALYST

There are two distinct categories into which user analysts may be classified. The first comprises those who have a non-computer background (perhaps in area studies, languages, the liberal arts) who use a technician to interact with the system. The other class of user analysts has a greater knowledge of ADP and can be expected to exploit the system to a greater degree than the first group. This latter group includes data specialists, file sponsors, and data base administrators.

Though both groups may benefit from the development of transparency oriented processing, it is probably the non-computer oriented analysts who may benefit most from increased user-orientation of data access. To enhance productivity and to utilize the analyst's problem-oriented training, the analyst cannot be unduly concerned with the mechanics of data retrieval or with the arbitrary differences between query languages, file structures, and data codes on a distributed network of diverse intelligence resources.

a. Analyst's Task

The intelligence analyst is responsible for assessing information related to critical and complex situations which have far-ranging political and military significance both nationally and internationally. The analyst frequently must work under considerable time pressure. To support his analysis and judgemental capabilities, a large volume of diverse information exists in multiple and uncoordinated sources. Analysts have traditionally developed highly personalized and often very sophisticated manual techniques for accessing information, based on experience and knowledge of sources. While the analyst may resort to rather sophisticated "shoe-box" techniques for storage of retrieved data, the main concern is not data storage or retrieval, but analysis.

The analyst's basic mission is to monitor and interpret intelligence information and report it in a timely and effective manner for further assessment at decision-making levels. In fulfilling this mission, he performs the following activities.

- Monitoring messages, indicators, and events
- Establishing and maintaining data files
- Seeking information from other analysts and data files
- Preparing finished intelligence reports
- Joint assessment of current intelligence situations
- Verifying and enhancing information through cross-correlation.

These activities are conducted within the context of three functional levels related to the urgency of the situation at hand. For the indications and warning (I&W) analyst these are:

- Watch Function - monitoring indicators and assessing current events
- Current Intelligence - actively tracking significant events and crisis situations
- In-depth Analysis - developing background information and analyzing information with long-term implications.

These functions are keyed to the operation of command centers and the formulation of precise military decisions based on the best possible intelligence information. Within this context, the analyst's objectives are the timely and accurate assessment and subsequent reporting of complete intelligence data. To accomplish these basic, but difficult, objectives the analyst has very real requirements which can be met through data processing support.

b. Data Analysis Requirements

There is a strong and continuing need for automated support to provide the analyst with ready access to data. A large measure of automatic data processing support is being developed using facilities already in place. However, these facilities are geographically dispersed and reside on independent hardware and software systems.

A major need is on-line access to data bases which are not part of the analyst's immediate system. To be useful to the analyst, ADP support must make minimal demand on him to learn access techniques. The ideal ADP support should unburden the analyst from an already substantial data processing workload. This requires the implementation of interactive, integrated networking concepts with sufficient translating and routing software to permit the analyst to access external systems in his own familiar terms. The provision of simplified, real-time access to remote data bases would greatly aid the analyst in cross-correlation, data assessment and reporting functions.

The volume and variety of intelligence data, and the ways in which it may be stored and handled in the intelligence community are great. Merely to monitor the vast array of information provided by sensors and other data collection devices within the analyst's area of specialization can be an immense job; the cross-correlation of data is staggering. Requirements for automatic data processing support clearly exist both with regard to the acquisition of data <u>and</u> its later manipulation for analytic purposes.

Data processing needs are critical to the nature of the indications and warning analyst's mission, functions, and activities. Automation can support not only the monitoring activities in which the analyst is frequently engaged, but will greatly assist the analyst's performance in crisis situations.

Analyst data needs amenable to the application of ADP technology generally focus on the access and manipulation of data. Both access and manipulation have several facets. Access requirements are broad and specific: the analyst has a great interest in a wide variety of sources and indicators, as well as specific information requirements. Since the analyst desires to keep abreast of developing situations, single elements of intelligence are sometimes critical; timeliness, validity, and reliability are also very important. The data manipulation needs of the analyst focus on the requirement to rapidly process and format large volumes of diverse information. By way of illustration, some access and manipulation needs are: immediate updating of key indicators, corroboration of hypotheses with initial intelligence, multi-source cross-correlation support, program generation for formatting unfamiliar data packages, real-time requests for specific information, and queries to experts outside of the analyst's field of specialization.

APPENDIX G

DESCRIPTION OF THE TIIN SYSTEM

1. INTRODUCTION

The purpose of this appendix is to provide a concise description of the function of each module included in the design of the Transparent Integrated Intelligence Network. The modules are in varying phases of development. These descriptions reflect the range of development stages and are, therefore, in varying degrees of detail.

The TIIN system has been divided into subsystems which perform discrete tasks. The functional descriptions which follow are organized according to subsystems.

Figure G-1 illustrates the TIIN components, and will be referred to throughout this appendix.

2. TERMINAL ACCESS PROCEDURES (TAP)

Components of the TAP subsystem appear on the left side of Figure G-1. The components are 1) the Data Request Command Interpreter, 2) the Data Request Dialog Processor, and 3) the Network Data Catalog Processor.

The TAP subsystem is designed to provide a convenient and natural user interface method. The functions of TAP are 1) to describe network data resources to a user/analyst, 2) to support the user in the construction of legal queries, and 3) to allow for rapid, interactive dialog with the network.

a. Data Request Command Interpreter

This processor will provide for a means of expressing a data request via a simple formal language consisting of simple verbs corresponding to Data Request Intermediate Format (DRIF) operators. The module will check the syntactic correctness of the user data request, and will produce the DRIF version to be sent to the Query Intermediate Processor (QIP).

b. Data Request Dialog Processor

This module provides a method for the analyst to interrogate network data bases without using a formal language. The

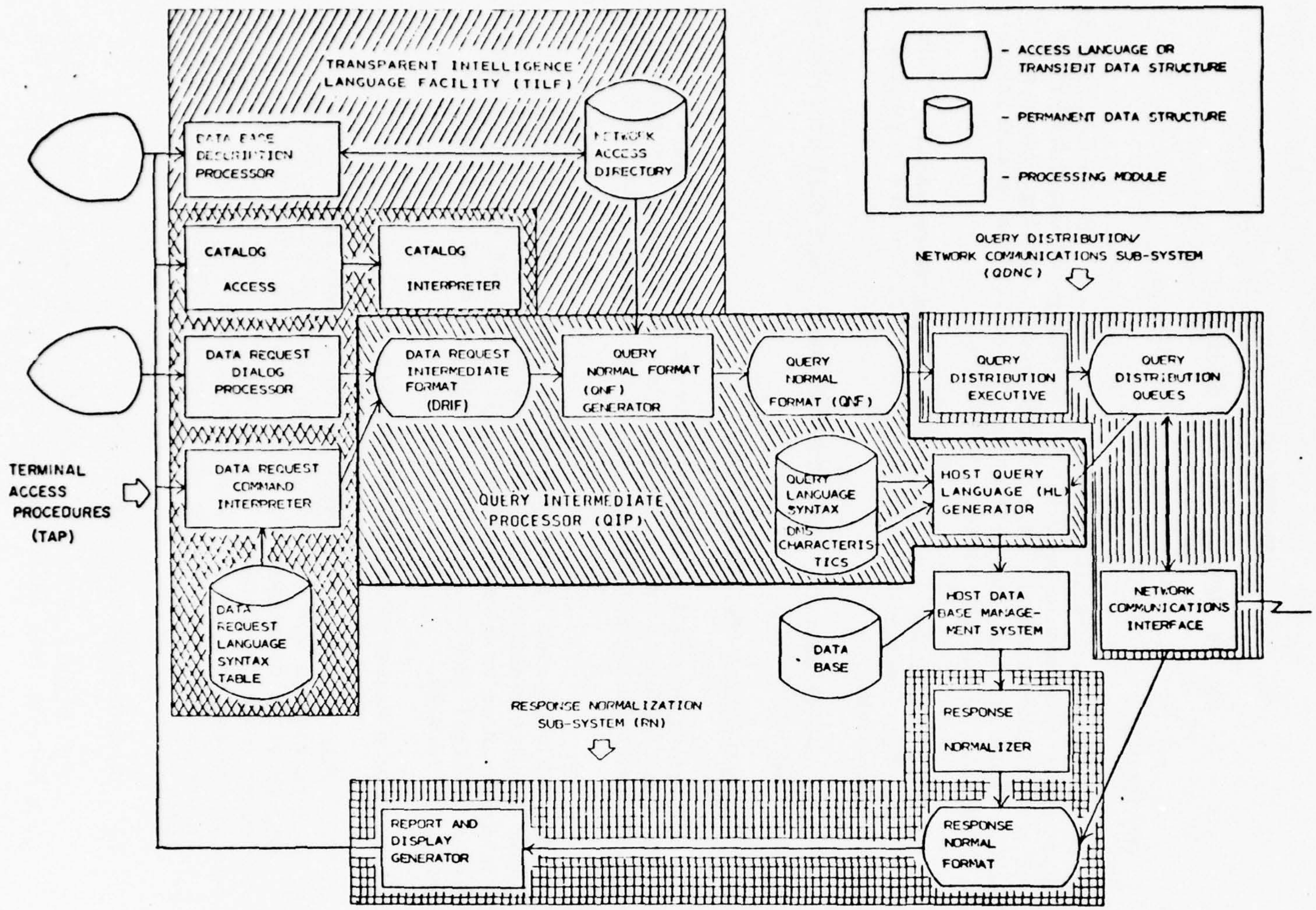


Figure G-1. Transparent Integrated Intelligence Network (TIIN) Relationship of Projects.

processor guides the analyst through data selection and output specification and translates the analyst's responses into queries phrased in DRIF. The module will provide information from the Network Access Directory to the analyst. Thus, if synonymous names are given for data elements, the system will feed back the standardized names, if requested. When the analyst does not specify data source files, the system will list possible sources and allow him to specify a choice or to set priorities. The system will further provide the anticipated response time for each file, and advise the analyst when the data was last updated.

c. Network Data Catalog Processor

A great deal of the descriptive information which is maintained in the Network Access Directory for system use is also useful to the analyst in identifying the precise data requirements. The Catalog Access displays the contents of the NAD in a way meaningful to the user. It also provides the data base administrator with the means to update the data base description statements relating to specific subject categories.

The catalog processor will display NAD information by subject category, prose descriptions of files related to subjects the analyst selects, prose descriptions of data elements belonging to selected files, and properties and attributes of selected elements.

The nature of the linkage between the catalog and the Network Access Directory is included in the catalog structural design to facilitate the interpretation function between analyst needs and NAD contents.

The catalog is to be implemented as a table driven process. No reprogramming will be required to accommodate data base changes.

3. TRANSPARENT INTELLIGENCE LANGUAGE FACILITY (TILF)

The components of the TILF subsystem appear in the upper part of Figure G-1. This subsystem is devoted to describing the contents of varied data bases to the network and keeping these descriptions up-to-date for real-time access from any node in the network. The subsystem includes the Data Description Processor and the Network Access Directory (NAD).

a. Network Access Directory (NAD)

The Network Access Directory (NAD) is the repository for all information required by the system about the contents of network data bases. Individual versions of the NAD will be created and maintained at each network node, and either all or part of each local NAD must be able to be easily transmitted to other network nodes on request. This feature will enable each node to periodically gather information from its own and other NADs, consolidate the information, and use it to locate data throughout the network.

The largest segment of a local NAD will be a file, each entry of which contains descriptive information about a local data element, a network standard element (which may or may not have a corresponding local element), or a local-usage name for a network data element. Other locally-maintained NAD files will contain descriptive information about the parent files and parent data bases of local elements and about legal values or ranges of values for local elements. The development of a network data catalog will add a subject index file to the NAD, and the development of the query distribution module requires the addition of routing information for remote data bases.

Data element entries must be accessible via two modes of user access. Normally, specific data elements will be "found" when an analyst expresses an interest in a particular subject; however, from frequent use of the network system, the analyst may choose to refer by name to elements, thus bypassing perusal of subject categories. Included in the information about a data element are:

- o Name
- o Element type (string or numeric)
- o Length and format information (if string)
- o Value range (if numeric)
- o Pointer to parent file
- o Pointer to parent data base
- o Pointer to network synonym

- o Pointer to value list
- o Host routing information

b. Data Description Processor

The Data Description Processor is one of the user interface modules of the TIIN network. Unlike the network query and retrieval functions available at the TIIN node, which may be used by any analyst who has access to the local computer facility, access to the data description function should be limited by password protection. Its use should be restricted to the authorized administrator of the local data base.

The Data Description Processor creates and updates the local NAD segment. Its function is to provide a means for the local data base administrator to describe to the network those elements in local data bases which may be made accessible to other network users. In addition to the initial description facilities, the Data Description Processor must provide update capabilities compatible with the real-time nature of network user requirements.

The mode of operation for the data description function should be interactive and guided by the Processor rather than the user. While the data base administrator is required to have more specialized knowledge than a user of the network, the focus of that knowledge should be on the structure and content of the data base rather than on complex system procedures.

4. QUERY INTERMEDIATE PROCESSOR (QIP)

QIP components appear in the center of Figure G-1. This subsystem deals with host selection, element name translation, and query translation. It is composed of two main parts, those modules which are user dependent, called the User QIP, and those modules which are host dependent, called the Host QIP.

The functions of the user QIP are 1) selection of host data bases and files containing the elements requested by the user, as indicated in the DRIF, and 2) translation of queries from DRIF into a network-wide standard format called Query Normal Format (QNF).

The host QIP translates the user query from the network standard format (QNF) into the host DBMS language.

The sequence of actions through which the query will be processed includes:

- o Identification of those files which contain the requested data elements
- o Substitution of network-standard element names for synonymous user element names
- o Creation of equivalent versions of a query for addressing different files which may contain similar data
- o Creation of sequences of queries when several files must be searched in sequence to retrieve data, or when several different search verbs are necessary to process all the search criteria
- o Reconciliation of distinctive functions in the QNF not provided for in the host language
- o Translation of network standard element names into element names in the host frame-of-reference
- o Generation of the host version of the user query.

5. QUERY DISTRIBUTION EXECUTIVE/NETWORK COMMUNICATIONS INTERFACE (QDNC)

QDNC components appear on the right side of Figure G-1. This subsystem deals with 1) routing of queries to data resources, both local and external, 2) tracking and logging of incoming and outgoing network communications, and 3) the initiation of transmission of data from local to external nodes.

a. Query Distribution Executive

This module tracks and routes queries and responses. For incoming queries, it makes an entry in the incoming query queue with status of "awaiting processing." The Distribution Executive polls the outgoing query queue for entries with "response received" status. Upon finding one, the Distribution Executive notifies the report and display function passing the message sequence number (MSN) of the response file.

b. Network Communications Interface

The Network Communications Interface initiates transmission of data from the local node to any other network node and receives data into the local node from other network nodes. The data which will pass through the Interface will consist of:

- o Queries being distributed to other network nodes for processing
- o Incoming responses from distributed queries
- o Queries coming into the node from other network nodes
- o Outgoing responses to queries from other nodes
- o Bulletins from one network to all other nodes to announce that a NAD update has taken place at the originating node
- o A request from one network node to a second node for transmission of the second node's NAD segment
- o A NAD segment transmitted from one network node to another.

Incoming and outgoing messages are blocked and routed using the communications modules of TISS. An outgoing "message" (any data transmissions between network nodes), is converted to a WCF (WICS Common Format) header block including the message sequence number (MSN), user identification, node identifier, message type (query, query response, NAD bulletin, etc.). Outgoing message traffic is handled by polling two queues: the outgoing query queue (for queries to be sent), and the incoming query queue (for responses to be sent). Incoming message traffic is also handled, with most message types being passed immediately to other network elements for processing.

6. RESPONSE NORMALIZATION (RN)

Components of the Response Normalization Subsystem appear in the bottom portion of Figure G-1. The Response Normalizer (RN), in

conjunction with the Response Normal Format (RNF), is designed to provide a capability for output standardization in the overall TIIN design. In general, the task of the RN is to provide the capability for converting data to a standard format after it has been retrieved from the local DBMS. Since data which is returned in response to a single query may be from separate files, it is likely to exhibit different characteristics with regard to coding conventions. The RN normalizes responses, converting them to a standard format which enables collation and reconciliation of conflicts and duplications. The RN also facilitates storage of data in the analyst's private file where it would be available for further processing (such as refining the original request, sorting, statistical analysis, graphic display, etc.).

APPENDIX H

LANGUAGE TRANSPARENCIES ACHIEVABLE VIA TIIN

1. INTRODUCTION

Underlying the need for transparency in query languages are the issues of user productivity and installation flexibility. As an analogy, a FORTRAN compiler makes machine code transparent to a programmer. The effect of this transparency is that it reduces the number of rules a programmer must unravel when writing and rewriting a program. Also, the installation has certain flexibilities: efficiency can be enhanced by improving the compiler, and costs can be reduced by changing to cheaper computers. Transparency *per se* is not enough: it is vital to use common interfaces (i.e., common languages) and common training to tap all the potential benefits.

Similarly, the user/analysts, the file designers, and the host DBMS sites are in a position to benefit from increased transparency and increased commonality of query languages. The following sections describe and illustrate several different language features. Some of these features serve to increase transparency, while other features serve to increase commonality between query languages. In either case, the purpose is to reduce the number of rules to which the user/analyst must conform. A hypothetical query language, called TEL, or Transparency Examples Language, will be used to illustrate several transparency features relative to a host query language such as DIAOLS.

2. VALUE DELIMITERS

Quotes (') will be used for value delimiters in TEL instead of colons (:) as in DIAOLS, or parentheses as in TILE, to illustrate that the value delimiter is a characteristic of the query language and is not fixed by the host DBMS. Also, numeric values may be expressed without delimiters. For example, 741231 would be the same as "741231" and would correspond to :741231: in DIAOLS or to (741231) in TILE.

In the QNF version of the user query the string of characters representing the actual value (without the value delimiters) is stored with a character count and a value type code indicating it is a user-specified value.

3. ONE GENERIC SEARCH VERB

A generic search verb (called RETRIEVE) will be used in TEL to illustrate that one verb can be used to express all the nuances of multiple specialized search verbs (REQUEST, TEXTSCAN, REFINE, RELATE, GEOGRAPHIC, CIRCLE, ROUTE) in a query language such as DIAOLS. The RETRIEVE verb in the TEL user language is converted at the user node by the user language processor to the code (SELECT) for a generic search

in the QNF internal language. The QNF version of the user query is host independent. After TIIN locates a host node to respond to the user query, the QNF version is converted to the query language of the host. In the QNF to host conversion the generic search may be converted to a combination of specialized searches.

4. INVERSION TRANSPARENCY

One generic search verb may be used to specify selection criteria for inverted data elements and non-inverted data elements, so that the use of different search methods is transparent to the user.

DIAOLS saves storage space and reduces update costs by not maintaining an inversion file for every element. The element inversions are relatively transparent since the same element name is used for either search method. The DIAOLS user has the option of taking advantage of the inversion file (by using the verb REQUEST or REFINE) or specifying a file scan (by using the verb TEXTSCAN).

For TEL the generic search verb (e.g., RETRIEVE) would be defined to use the inversion file whenever possible and otherwise would use a file scan. TIIN/QIP would partition the selection phrase into two selection phrases: one for a REQUEST verb, to generate a hit file, and the remaining selection criteria would be processed via a TEXTSCAN verb which would examine only the records indicated by the hit file. For example, assume B and C are inverted elements, but A and D are not in the following query.

```
TEL:     RETRIEVE A EQ 'AVAL' AND B EQ 'BVAL' AND (C GT 'CVAL' OR D
         LT 'DVAL');
```

would be re-expressed as

```
DIAOLS:  REQUEST B EQ :BVAL:.
         TEXTSCAN A EQ :AVAL:
         AND (C GT :CVAL: OR D LT :DVAL:).
```

Note that the phrase (C GT :CVAL: OR D LT :DVAL:) can be included in the selection criteria of the request verb only if both C and D are inverted elements.

This partitioning of a selection phrase is essentially what a user/analyst must be concerned with. Though the rules are simple, the time and effort involved in learning the rules and conforming with the rules results in less productivity for every user/analyst. Another consideration is the impact on the file design: due to changing demands for information it may be desirable to invert an element or delete the inversion file for an element, but the flexibility to make the change is somewhat restricted when consideration is given to the impact on the collection of canned queries and the impact on the user/analysts. There is more flexibility in file design and more productivity for user/analysts when the choice of search method is handled automatically.

The flexibility and productivity are achieved even though less efficient query processing might result in some special situations.

5. STRUCTURE TRANSPARENCY

The distinction between "independent conjunction" and "related conjunction" can be expressed using different logical operators (e.g., AND vs. WITH) under one generic search verb.

The distinction between independent conjunction and related conjunction is expressed in DIAOLS by using different search verbs. The only vehicle for expressing related conjunction is the verb RELATE, while the other search verbs (REQUEST, TEXTSCAN, REFINE) always process a conjunction as an independent conjunction. For example, if a user wanted to select the ships which arrived in Baltimore before July 11, 1976, and the file on ships has PORT and ARRIVAL-DATE as relational periodic (i.e., level 2) elements, the query must be expressed as follows:

```
DIAOLS:  RELATE PORT EQ :BALTIMORE:
         TO ARRIVAL-DATE LT :760711:.
```

If the analyst had used any other retrieval verb with the same criteria, the operator AND would be used to conjunct the criteria, and the result would be independent conjunction, as follows:

```
DIAOLS:  TEXTSCAN PORT EQ :BALTIMORE:
         AND ARRIVAL-DATE LT :760711:.
```

For this query the use of independent conjunction would result in a hit file containing the ships which were in Baltimore (at some time) and had an arrival date before July 11, 1976 (at some port, not necessarily Baltimore). The effect of independent conjunction is the same as the effect of two consecutive queries (one query for each criterion), for example:

```
DIAOLS:  TEXTSCAN PORT EQ :BALTIMORE:
         TEXTSCAN ARRIVAL-DATE LT :760711:.
```

The effect of independent conjunction would be retrieval of all the ships in the sample file listed in Appendix E, whereas the effect of related conjunction would be retrieval of only two ships (the GORCH FOCK and the EENDRACHT).

The TIIN system supports a special logical operator WITH to express related conjunction, while independent conjunction is expressed using AND, as for example:

```
TEL:     RETRIEVE PORT EQ 'BALTIMORE'
         WITH ARRIVAL-DATE LT 760711;

TEL:     RETRIEVE PORT EQ 'BALTIMORE'
         AND ARRIVAL-DATE LT 760711;
```

The operator WITH would have the same interpretation as AND if the conjuncted criteria applied to elements which are not actually related in the data base. This allows the user/analyst to express queries without knowing the structure of the host data base. This factor might be useful when a query is forwarded to several different data bases, possibly with different structures.

6. GEOGRAPHIC OPERATOR TRANSPARENCY

All relational operators available at host systems may be specified under the scope of one generic search verb.

The user might specify a geographic search by using an area designator and a relational operator such as INSIDE, OUTSIDE, or ALONG. Such selection criteria might also be combined with other selection criteria via logical operators such as AND, OR, NOT. For example:

```
TEL:     RETRIEVE STATUS EQ 'ACTIVE'
         AND INSIDE POLY-1;
```

This would correspond to a query such as the following:

```
DIAOLS:  TEXTSCAN STATUS EQ :ACTIVE:.
         GEOGRAPHIC INSIDE POLY-1.
```

The details of specifying the perimeter points have been omitted here for clarity. In DIAOLS the operators INSIDE and OUTSIDE denote a relation between a user-specified area and an implicit operand, the locational data in the record.

7. BOOLEAN COMPLETENESS

Selection criteria (e.g., GRADE GT 5) may be thought of as Boolean expressions (i.e., as binary-valued expressions which are either "true" or "false"), and as such may be used to form complex Boolean expressions from operators such as AND, OR, NOT, with parentheses used to indicate relative grouping of expressions. This is a language feature borrowed from elementary logic: if A and B denote Boolean expressions the NOT A, (A), A AND B, and A OR B are also Boolean expressions. Since many user/analysts have studied elementary logic either directly or indirectly, they have a "built-in" ability to read and write Boolean expressions. This language feature is supported in the TIIN system by automatically transforming the query into an equivalent Boolean expression to conform with the different restrictions of each host. DIAOLS allows NOT *only before* a relational operator such as GT, LT or EQ. For example:

```
TEL:     NOT (B EQ 'BVAL' OR C GT 'CVAL').
```

would be re-expressed by TIIN as

```
DIAOLS:  B NOT EQ :BVAL: AND C NOT GT :CVAL:.
```

8. RELATIONAL COMPLETENESS

The arithmetic relational operators include LT, LE, GT, GE, EQ, NE and may be used in the context of a relational operator in a relational expression whenever there is a valid semantic interpretation for the resulting expression. This language feature is common to most query languages. The six arithmetic relational operators are included in the TIIN internal representation (QNF) for user queries to achieve host independence. For the DIAOLS host the operators LE, GE, NE are not accepted directly and must be re-expressed as NOT GT, NOT LT, NOT EQ. This restriction would be transparent to any user of a TIIN supported user language. For example, if a user happens to think of "after 1974" as "GE 750101" rather than "GT 741231", either would be valid grammatically.

TEL: ARRIVAL-DATE GE 750101

would be accepted and would be re-expressed as

DIAOLS: ARRIVAL-DATE NOT LT :750101:

A special check is necessary if the quantifier ALL is involved.

TEL: ARRIVAL-DATE ALL GE 750101

must be re-expressed as

DIAOLS: ARRIVAL-DATE NOT LT :750101:

rather than simple replacing "GE" by "NOT LT" as in

DIAOLS: ARRIVAL-DATE ALL NOT LT :750101:

since that would be processed the same as

DIAOLS: ARRIVAL-DATE LT :750101:

because ALL NOT and NOT ALL are treated the same and are both equivalent to a null-phrase. These special checks and host restrictions would be handled automatically without involving user attention.

9. ENGLISH ACRONYMS

The English phrase equivalent of the relational operators may be used in the context of a relational operator in a relational expression. The general premise underlying this language feature is that there shall be readable synonyms for the reserved words in the query language. Such features require no special processing in the TIIN system since phrase equivalence is accomplished by translating the phrase into the same internal QNF code. Thus, phrases such as LT, LESS, IS LESS THAN, etc. can be given the same internal encodement and thereby will have the same interpretation. Similar synonyms are available in most query languages, including DIAOLS.

10. DETERMINERS

Determiners such as EVERY, SOME, NO may be specified as prefix modifiers for element names, in analogy with the usage of determiners in English. Generally such determiners are used in query languages to indicate that a selection criterion must be applied to all occurences of an element under a selected root node. Other examples of determiners include AT LEAST n, MAX, LAST, FIRST, LATEST, AVERAGE, etc.

For TEL-to-DIAOLS queries this language feature would involve moving the quantifier from the element to the relational expression. NO would change to NOT, EVERY to ALL, SOME to a null operator, while AT LEAST, MAX, LAST, etc. would be rejected. For example:

TEL: NO ARRIVAL-DATE LT 750101

Would be re-expressed as

DIAOLS: ARRIVAL-DATE NOT LT :750101:

while

TEL: EVERY ARRIVAL-DATE LT 750101

would be re-expressed as

DIAOLS: ARRIVAL-DATE ALL LT :750101:

APPENDIX J

MULTI-FILE QUERIES

1. PURPOSE

The purpose of this appendix is to define the concepts of multi-file queries, to discuss the development stages for such a facility within the framework of TIIN, and to specify an expanded version of the DRIF/QNF structure capable of handling multi-file queries.

2. GENERAL DESCRIPTION OF FULL CAPABILITY

The development of a multi-file query capability in TIIN is an advanced step in the development of an integrated intelligence network. A fully operational multi-file query facility will enable an analyst to consider a network of intelligence data bases as a single distributed data base, accessible through a single query language and user protocol. In such an environment the modules of TIIN will extract all data in the network that is relevant to an analyst's query and present it to the analyst as a combined report containing all relevant information available anywhere in the network.

3. REQUIRED TIIN ALGORITHMS

The design of a complete multi-file query capability and the resulting integrated network data base depends on the development of several capabilities within TIIN. This section describes the algorithms required in each TIIN subsystem in order to obtain a full multi-file capability.

a. QIP Capabilities

The following capabilities must be added to the prototype design of QIP in order to achieve a full multi-file query capability.

(1) Multiple Host Selection

The prototype host selection algorithm in QIP selects a single host file that may be capable of answering the query. The next development step is a selection algorithm capable of locating all files in the network which may contain a response to the query. Such an algorithm will guarantee that all files in the network containing all requested data elements will be queried.

(2) Multiple QNF's

The prototype QNF generator module creates a single QNF structure out of each acceptable DRIF structure that it receives. In order to query files at several different data bases, several versions of the query must be created and the resulting QNF's routed to different network nodes.

(3) Query Image Creation

Whenever several parallel network processes are initiated as a result of a single DRIF, TIIN at the user node must retain information concerning the relationship between the processes. Such information, referred to as the Query Image (QI), is created by the QNF Generator as a by-product of QNF generation, and is communicated to the QDNC for future reference by other TIIN modules.

(4) Query Decomposition

At times, the combined responses from several files may form an answer to an analyst's query, while none of the files alone contain the necessary information. Each of the queries submitted to the data bases is called a partial query, since it only contains a portion of the original request. An advanced QIP algorithm will be developed to spot cases where partial queries will provide the answer to a problem, and to decompose user queries into partial queries, at the same time specifying the rules for combining the resulting partial responses.

(5) Hit File Processing

A special case of partial queries are partial queries processed sequentially, with a hit file created by the first partial query, and incorporated into the next partial query. A QIP algorithm will be developed to incorporate a hit file into the conditional table of a QNF according to specifications in the QI.

b. QDNC Capabilities

The following QDNC algorithms will be developed in order to allow for multi-file query processing.

(1) Query Image Maintenance

It is the function of QDNC to create, maintain, and access the Query Images of all queries originating at the node. The Query Image is a tree structure, corresponding to a single user request (DRIF). Each terminal node in the structure describes a QNF resulting from the original DRIF; each non-terminal node in the structure describes the relationship between and rules for combining the results of several QNF's. A terminal node may contain a QNF and, optionally, an RNF resulting from the QNF.

(2) QNF Routing

Based on the Query Image, the QDNC sends out QNF's and receives RNF's. This feature becomes especially significant in combination with sequential queries, where the initiation of some QNF's is triggered by the arrival of a response to a preceding QNF.

(3) Response Synchronization

Whenever one user request (DRIF) results in multiple QNF's, the data base responses can be expected to arrive at the originating user node at different times. It is the job of QDNC to synchronize the responses by retaining the early responses until all responses have been received, and then presenting the user RN with all extracted data for report generation.

c. RN Capabilities

The following Response Normalization subsystem capabilities are an intrinsic part of the overall TIIN multi-file capability.

(1) Query Image Interpretation

User RN must communicate with QDNC to obtain normalized data and instructions for combining the retrieved data files.

(2) Data Management

User RN must be capable of performing several data management functions, such as sorting, merging, and matching sequential files of records. These operations are primitives necessary to combine responses obtained from different data bases in order to present the user with a single comprehensive responses.

4. DEVELOPMENT STAGES OF MULTI-FILE QUERY FACILITY

User capabilities in the area of multi-file queries can be described as a progression of development stages, each offering new user capabilities and dependent on additional TIIN facilities. The realationship between TIIN facilities and the resulting user capabilities is shown in Figure J-1. This section describes in detail the user capabilities offered at each stage.

a. Parallel Access to Multiple Files (Single Host)

At this stage, QIP will locate all relevant files in the network and submit the query to a single data base with the largest number of qualifying files, requesting responses from all such files in the data base.

b. Parallel Access to Multiple Hosts

At this stage, QIP will locate and query all qualifying files in the network. If several qualifying files or data bases are found, the analyst and QIP will not have to choose which one to query, since all will be queried automatically. The resulting responses will be normalized, merged, duplicates removed, and a single, comprehensive report will be presented to the user.

c. Partial Queries

At this development stage, the user will have the option of specifying a set of partial queries, together with instructions on merging the resulting partial responses. TIIN will query all specified files and data bases with the partial queries, and merge the partial responses into a single report.

d. Automatic Partial Queries

This stage will automatically create partial queries. QIP will analyze a query and the appropriate files in the network. If partial queries seem likely to produce results, the query will automatically be split into several partial queries and then be processed as specified above. An analyst will therefore not need to concern himself with any details of file selection, and will always be guaranteed to receive all relevant information available from the network.

USER CAPABILITIES

1. Parallel Access to Multiple Files (Single Host)
2. Parallel Access to Multiple Hosts
3. Partial Queries
4. Automatic Partial Queries
5. Sequential Queries

TIIN FACILITIES

QIP					
1. Multiple Host Selection	X	X	X	X	X
2. Multiple QNF's		X	X	X	X
3. Query Image Creation		X	X	X	X
4. Query Decomposition				X	X
5. Hit File Processing					X
QDNC					
1. Query Image Maintenance		X	X	X	X
2. QNF Routing	X	X	X	X	X
3. Response Synchronization		X	X	X	X
RN					
1. Query Image Interpretation		X	X	X	X
2. Data Management			X	X	X

Figure J-1. Relationship Between Implementation of Advanced TIIN Facilities and Resulting User Capabilities.

e. Sequential Queries

Sequential queries are defined as series of queries, each query dependent on a hit file produced by the previous query. Since sequential queries are logically equivalent to partial queries, TIIN will not automatically introduce sequential queries. At this stage, however, the user will have the capability to explicitly request sequential queries, and TIIN will have the capability to process such queries.

METRIC SYSTEM

BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

SUPPLEMENTARY UNITS:

Quantity	Unit	SI Symbol	Formula
plane angle	radian	rad	...
solid angle	steradian	sr	...

DERIVED UNITS:

Quantity	Unit	SI Symbol	Formula
Acceleration	metre per second squared	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	siemens	S	A/V
electric field strength	volt per metre	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm		V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	candela per square metre	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	joule per kilogram-kelvin	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
1 000 000 000 000 = 10^{12}	tera	T
1 000 000 000 = 10^{9}	giga	G
1 000 000 = 10^{6}	mega	M
1 000 = 10^{3}	kilo	k
100 = 10^{2}	hecto*	h
10 = 10^{1}	deka*	da
0.1 = 10^{-1}	deci*	d
0.01 = 10^{-2}	centi*	c
0.001 = 10^{-3}	milli	m
0.000 001 = 10^{-6}	micro	μ
0.000 000 001 = 10^{-9}	nano	n
0.000 000 000 001 = 10^{-12}	pico	p
0.000 000 000 000 001 = 10^{-15}	femto	f
0.000 000 000 000 000 001 = 10^{-18}	atto	a

* To be avoided where possible.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

AMERICAN REVOLUTION BICENTENNIAL
1776-1976

*U.S. GOVERNMENT PRINTING OFFICE 1977-714-025/132